DTIC FILE COPY

①

AD-A230 499

DTIC
ELECTE
JAN 08 1991
D

Speech Coding and Compression Using Wavelets and Lateral Inhibitory Networks

THESIS

Richard Ricart
Captain, USAF

AFIT/GE/ENG/90D-51

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

91 1 3 061

①

AFIT/GE/ENG/90D-51

DTIC
ELECTE
JAN 08 1991
S D
D

Speech Coding and Compression Using Wavelets
and
Lateral Inhibitory Networks

THESIS

Richard Ricart
Captain, USAF

AFIT/GE/ENG/90D-51

Approved for public release; distribution unlimited

AFIT/GE/ENG/90D-51

Speech Coding and Compression Using Wavelets
and
Lateral Inhibitory Networks

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Richard Ricart, B.S.E.E.
Captain, USAF

December 1990

Accession For
NTIS GRA&I ✓
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Approved for public release; distribution unlimited

QUALITY INSPECTED 4

Preface

The purpose of this ~~paper~~ Thesis is to introduce the concept of lateral inhibition as a generalized technique for compressing time/frequency representations of electromagnetic and acoustical signals, particularly speech. This requires at least a rudimentary treatment of the theory of frames—which generalizes most commonly known time/frequency distributions—the biology of hearing, and digital signal processing. As such, this material, along with the interrelationships of the disparate subjects, is presented in a tutorial style. This may leave the mathematician longing for more rigor, the neurophysiological psychologist longing for more substantive support of the hypotheses presented, and the engineer longing for a reprieve from the theoretical barrage. Despite the problems that arise when trying to appeal to too wide an audience, I hope this thesis is a cogent analysis of the compression of time/frequency distributions via lateral inhibitory networks.

I want to thank my thesis committee for their guidance, which was instrumental in helping me cohere the topics investigated. Individual thanks to Col. David Norman for his immediate and helpful feedback regarding questions in digital signal processing, Dr. Bruce Suter and his colleague Dr. Mark Oxley for the hours they devoted in reviewing the wavelet/frames mathematical literature with me, and to Maj. Steve Rogers for his superb insight and his ability to immediately pick out the weaknesses and strengths of the research as it progressed. Most importantly, my deepest and most sincere thanks to my advisor, Dr. Matthew Kabrisky, which has made my AFIT experience so rewarding. Without his broad knowledge in biology and engineering this research effort would not have been the success I feel it is. I have truly enjoyed our many lively discussions and *gedanken* experiments. Final thanks to my friends Capt. Dennis Ruck and Capt. Greg Tarr for sharing their expertise in Unix and C programming with me, and to Pam Young for her administrative assistance and her cheerful disposition.

Richard Ricart

Table of Contents

Page

Preface ii

Table of Contents iii

List of Figures vi

List of Tables ix

Abstract x

I. Introduction 1
  - Background 1
  - Problem Statement 3
  - Assumptions 4
  - Scope 4
  - Standards 4
  - Equipment 5
  - Approach 5
  - Summary 6

II. Literature Review 7
  - Introduction 7
  - Background 7
  - The Short-Time Fourier Transform 9
  - The Gabor Representation 11
  - Wavelets 14

Page

Introduction 14
Affine Wavelet Transform 15
Biological Considerations 19
Introduction 19
The Auditory Periphery 19
Lateral Inhibition Networks 23
Summary and Final Comments 26

III. System Design 27
Introduction 27
System Environment 27
Definitions and Notation 28
Short-Time Fourier Decomposition/Reconstruction 29
Introduction 29
The Short-Time Discrete Fourier Transform 30
Gabor Decomposition/Reconstruction 38
Affine Wavelet Decomposition/Reconstruction 42
Compression Using Lateral Inhibition 47
Review of Fourier Spectrum Compression Methods 47
Review of Affine Wavelet Spectrum Compression Methods 48
LIN Design 49
Summary 58

IV. Results 59
Introduction 59
Results Based on Short-Time Fourier and Gabor Spectra 60
Results Based on Affine Wavelet Spectra 94
Noise Filtering Analysis 104
Summary of Results 104

Page

V. Conclusions and Recommendations 106

Introduction 106

Summary of Results, Conclusions, and Recommendations 106

Biological Implications 109

Appendix A. Code for Gabor Transform 113

Appendix B. Code for Short-Time Fourier Transform 119

Appendix C. Code for Reconstruction Algorithm from Short-Time Fourier Spectrum 127

Appendix D. Code for LIN Compression of W-H Spectra 133

Appendix E. Code for Haar Wavelet Decomposition, Compression, and Reconstruction 139

Appendix F. Code for Morlet Wavelet Decomposition, Compression, and Reconstruction 149

Bibliography 160

Vita 165

List of Figures

Figure Page

1. The Gaussian Window and its Biorthogonal Function 13
2. The Exponential Window and its Biorthogonal Function 13
3. The Gabor Time/Frequency Analysis 14
4. The Haar Mother Wavelet 16
5. The Haar Wavelet at Four Resolutions 17
6. The Affine Wavelet Time/Frequency Analysis 18
7. Filter Model of the Cochlea 22
8. Lateral Inhibition Network Model 24
9. The Rectangular Window 31
10. The Compactly Supported Gaussian Window 32
11. The Hamming Window 32
12. Effect of Windowing with non-Rectangular Functions 34
13. Sum of Gaussian Windows with 50% Overlap 35
14. Sum of Gaussian Windows with 25% Overlap 36
15. Sum of Hamming Windows with 50% Overlap 36
16. The Morlet Wavelet 44
17. Lateral Inhibition Network Model 51
18. Typical Mexican Hat Filter Function 53
19. ERB and CB as Functions of Frequency 54
20. LIN Frequency Bandwidth and ERB vs Frequency 56
21. LIN Frequency Bandwidth and CB vs Frequency 57
22. Original Signal and Reconstructed Signal from Spectrogram Using a Rectangular Window 63
23. Difference Signal Between Original and Reconstructed from Spectrogram Using a Rectangular Window 64

Figure Page

24. Original Signal and Reconstructed Signal from Spectrogram Using a Hamming Window 65

25. Difference Signal Between Original and Reconstructed from Spectrogram Using a Hamming Window 66

26. Original Signal and Reconstructed Signal from Spectrogram Using a Gaussian Window 67

27. Difference Signal Between Original and Reconstructed from Spectrogram Using a Gaussian Window 68

28. Original Signal and Reconstructed Signal from Gaussian Gabor Spectrum 69

29. Difference Signal Between Original and Reconstructed from Gaussian Gabor Spectrum 70

30. Original Signal and Reconstructed Signal from Short Gaussian Gabor Spectrum 71

31. Difference Signal Between Original and Reconstructed from Short Gaussian Gabor Spectrum 72

32. Original Signal and Reconstructed Signal from Gabor Spectrum Using an Exponential Window 75

33. Difference Signal Between Original and Reconstructed from Gabor Spectrum Using an Exponential Window 76

34. Exponential Gabor Decomposition/Reconstruction Example of Aligned Signal in Time and Frequency 77

35. Exponential Gabor Decomposition/Reconstruction Example of Unaligned Signal in Time and Frequency 78

36. Frame 9 of Spectrogram for Each Window 79

37. Frame 9 of Gabor Spectrum for Each Window 80

38. Compression of Spectrogram Using ERB Rule and Low Threshold 82

39. Compression of Spectrogram Using CB Rule and Low Threshold 83

40. Compression of Spectrogram Using CB Rule and High Threshold 84

41. Compression of Gabor Spectrum Using ERB Rule and Low Threshold . . 85

42. Compression of Gabor Spectrum Using CB Rule and Low Threshold . . 86

Figure Page

43. Compression of Gabor Spectrum Using CB Rule and High Threshold . . 87
44. Aliasing Effects Obtained from Compressed Spectrogram Using a Hamming Window 88
45. Aliasing Effects Obtained from Compressed Gabor Spectrum Using a Gaussian Window 89
46. Reconstructed and Difference Signal from Compressed Gabor Spectrum Using a Gaussian Window 91
47. Reconstructed and Difference Signal from Compressed Gabor Spectrum Using a Gaussian Window with Truncated Biorthogonal Function 92
48. Reconstructed and Difference Signal from Haar Wavelet Coefficients . . 96
49. Reconstructed and Difference Signal from Morlet Wavelet Coefficients . 97
50. Reconstructed and Difference Signal from Narrow Variance Morlet Wavelet Coefficients 98
51. Levels 0–2 of Frame 9 of Haar Wavelet Decomposition 99
52. Levels 3–5 of Frame 9 of Haar Wavelet Decomposition 100
53. Levels 6–7 of Frame 9 of Haar Wavelet Decomposition 101
54. Levels 3–5 of Frame 9 Nearest Neighbor LIN Compression Result 102
55. Levels 6–7 of Frame 9 Nearest Neighbor LIN Compression Result 103
56. Respective Spectra of Original and Reconstructed Signals 112

List of Tables

Table Page

1. Frequencies Analyzed Using Haar and Morlet Wavelets 46
2. Bandwidths Analyzed by LIN 55
3. Intervals Analyzed by LIN 58
4. MSE of Reconstructed Signals 74
5. Compression Ratios Obtained from Short-Time Fourier Spectrum 93
6. MSE of Reconstructed Signals from Compressed Spectra 93
7. MSE of Reconstructed Signals From Wavelet Coefficients 95

AFIT/GE/ENG/90D–51

Abstract

A generalized method of compressing speech using lateral inhibition networks (LINs) is proposed in this thesis. Speech signals are first decomposed using three time/frequency transforms: the short-time Fourier transform, the Gabor transform, and the affine wavelet transform. Redundant time/frequency coefficients are then automatically eliminated using the dynamics of a LIN. The speech is finally resynthesized from the compressed representations and tested for intelligibility. The LIN is modeled as a system of shunting non-linear differential equations in the form of the neuronal cell membrane equations first found by Hodgkin and Huxley. Thus, the LIN can be described as a neural network. LINs perform several functions; the most important being contrast enhancement. In contrast enhancement, spatial peaks and edges as well as temporal changes are highlighted. The best results were obtained from the compressed short-time Fourier spectrum. Nearly 30 times compression—which relates to over 95% *elimination* of the spectrum—resulted in clearly intelligible speech. However, elimination of only a small percentage of the affine wavelet spectrum produces wide band noise that is offensive to the hearing mechanisms.

Speech Coding and Compression Using Wavelets
and
Lateral Inhibitory Networks

I. Introduction

Background

The primary method of communication between humans is speech. Speech is so natural that people have demanded from science and technology the means by which the human voice can be transmitted in communication systems. As far as the technology is concerned, easier, more efficient communication methods are possible—Morse-code transmission for instance. However, these alternative methods require practice and training by the user. It is much more convenient for a person to pick up a telephone and simply talk rather than tap codes on a key. With increasing reliance on machines, particularly computers, the demand for natural communication systems between machines and their operators has grown as well.

The demands for natural speech communication systems have created specific speech processing applications. Some of the most common applications include speech compression, for more efficient transmission and storage of the speech signal; speech synthesis, for voice response systems; speech recognition; speaker recognition and verification; speech processing aids for the hearing impaired; and speech enhancement of noisy or mutilated speech [4, 38, 49, 53, 58].

The fundamental question in speech processing, regardless of the application is: how is the speech signal to be characterized or represented? How the speech signal is represented is crucial, for that representation must contain the information conveyed by the speaker in a way that can be extracted by the given system. Rabiner and Schafer proposed two major considerations in a speech processing system:

> 1. Preservation of the message content in the speech signal.
> 2. Representation of the speech signal in a form that is flexible so that modifications may be made to the speech signal without seriously degrading the message content. [58:2]

These considerations are the cruces of this research.

The question of representation takes on the greatest meaning in automatic speech recognition. After nearly half a century of research, the recognition of natural speech is still mostly an unsolved problem. In a recent review, Lippmann [42] reported that the best recognizers are able to achieve only 50% sentence accuracy. Because of the slow progress in resolving the major problems in speech recognition, some researchers are questioning the traditional speech representation schemes [55]. These traditional schemes revolve around the short-time Fourier transform, and Linear Predictive Coding (LPC) [53, 58]. Instead, calls for studying more natural speech encoding methods are growing.

Recently, researchers have studied the output of inner ear models as feature sets (representations) for recognition systems [42, 55]. Lippmann [42] reported initial comparative studies that show biologically motivated preprocessors are significantly increasing the performance of current recognizers, particularly in the presence of noise. However, the inner ear models are very computationally intensive and are not useful for practical systems.

In this thesis, an alternative to a full inner ear model is examined. This alternative is the compressed wavelet representation via a nonlinear lateral inhibitory network (LIN) (see Chapters II and III). Briefly, the wavelet representation of speech—indeed, of an arbitrary signal—may be defined as the summation of a finite number of weighted *wavelets* or *elementary signals* shifted in time and frequency, or shifted in time and dilated—depending on the class of wavelet [66]. The wavelets are defined locally in time and modulated; thus, this representation describes the local frequency variations of a signal.

Besides sharing important characteristics with biological models, the wavelet representation can be computed using algorithms of complexity comparable to the fast Fourier transform (FFT). Therefore, this representation has the potential of providing the benefits of inner ear model feature sets at a significantly reduced computational cost. Furthermore, LINs provide additional characteristics, such as automatic gain control (AGC), enhancement of signals in noise, and substantial information reduction capabilities that may prove useful in the coding and compression of speech.

Recent mathematical advances have generalized a wide range of time/frequency analyses by means of wavelets and their associated frames. Frames and frame operators, along with specific examples, are defined in the following chapter. For now it is sufficient to think of these as an umbrella description for time/frequency representations. One of the main goals of this research is to match this general description of signals—speech or otherwise—with a general approach to compressing these representations based on proven biological methods. Indeed, the methods developed here for compressing three specific wavelet representations of speech can be generalized to compress any wavelet transform space defined under the Weyl-Heisenberg and affine classes. These methods can be applied to analyzing and coding a wide range of electromagnetic and acoustic phenomena.

Problem Statement

The purpose of this research is to develop non-linear lateral inhibitory networks for coding and compressing the affine and Weyl-Heisenberg wavelet representations of speech. The compression schemes developed are also tested for enhancing noisy speech.

Assumptions

In order to perform the analyses described above, all speech signals are sampled at 8 kHz. Energy in speech drops at the rate of 6 dB/octave above 600 Hz and there is little above 4 kHz. Thus, 8 kHz is a usable sampling rate.

The speech signals are recorded in a functioning signal processing laboratory without guarding against background noise. In addition, additive white Gaussian noise (AWGN) is used to corrupt the speech signals in the noise filtering analysis.

Scope

The vocabulary used in this thesis is the spoken digits, zero through nine. In addition, the phrase

Hot today; chilly and cold tomorrow

is used to test the effects of continuous speech. This phrase is chosen because of the good balance between the fricatives and the vowel sounds found in the words.

Standards

The results of compressing the speech signals in this effort are compared against the results of similar research conducted at New York University, Courant Institute of Mathematical Sciences, and at the Air Force Institute of Technology (AFIT) using more conventional speech compression schemes.

The data compression and noise filtering analyses are compared against the analyses of Alenquer [1], Bashir [4], Kabrisky et al. [38], and McMillan [49]. The research cited assumes that speech can be coded using a relatively small number of spectral components [48, 57]. The spectral components were found via the short-time Fourier transform using a Hamming window with 50% overlap . The spectral components were then truncated using a rule-based system. Thus, the comparison is between different representation schemes as well as different compression schemes—a rule-based system versus a dynamical LIN system. A final comparison is made between the compression results of this research and those of Mallat and Zhong [45]. In that study, the investigators were able to completely reconstruct signals from only the local extrema of the affine wavelet representation.

Equipment

All software development is performed on a NeXT computer. This machine has a resident signal processor, the Motorola 56001 Digital Signal Processor, which is used for inputting, digitizing, and processing all voice signals.

The NeXT's built-in microphone jack is connected to an analog-to-digital converter known as the CODEC (coder-decoder). The CODEC uses a 8012.8 Hz sampling rate and 8-bit μ-law non-uniform quantization.

Approach

The 8 kHz sampled speech is decomposed into time/frequency components using three different wavelet transforms. These are the short-time Fourier transform, the Gabor transform, and the affine wavelet transform. Although not usually considered as wavelet transforms, the short-time Fourier transform and the Gabor transform are special cases of wavelet operators in Weyl-Heisenberg frames [31, 32] (see Chapter II for further details).

After decomposition, the respective wavelet time/frequency components of each signal are then compressed by the LIN. It is then determined whether these compressed representations can yield intelligible speech after reconstruction. If so, the results produced from each representation are compared. These results are also compared to the results found in [4, 45, 49]. Bit rates are then estimated for each representation. This rate is compared to the rate determined in [49] and current state-of-the-art bit rates. Finally, the speech signals are corrupted by AWGN, processed, compressed, and reconstructed. The reconstructed signals are tested for intelligibility and the signal-to-noise ratio is established.

Summary

This introductory chapter has outlined the main problem addressed in this research effort. In short, this research investigates a general method of compressing wavelet frame spaces using biologically inspired lateral inhibition networks. More specifically, the combined wavelet transform/LIN process is investigated for compressing and reducing noise in speech signals. A brief description of the approach, the scope, and the standards used was also given. In Chapter II, the wavelet frame is defined in the more exact language of mathematics. Preceding this definition, a brief background discussion is provided of several well known time/frequency analysis schemes, including the short-time Fourier transform and the Gabor transform, and how they fit in the wavelet frame taxonomy. Finally, the chapter concludes with the biological considerations that are most relevant to this research. Chapter III describes of the methodology and the system design. Chapter IV discusses the results and the comparisons to similar research. Finally in Chapter V, the conclusions and recommendations are given.

II. Literature Review

Introduction

This chapter begins with a brief historical perspective of a number of time/frequency analyses. Their interrelationship and their classification under the more general wavelet frames is also given. The short-time Fourier transform, the Gabor transform, and the affine wavelet transform are then defined. The discussion then turns towards the biological. Some putative dynamics of the inner ear and LINs are introduced. These biological considerations are important because they highlight the shared computational analysis performed by the inner ear and operators in affine frames. Furthermore, these biological considerations play a fundamental role in the design of the system used in this research. The full connection between the biology and affine frames, and how this connection influences the system design is found in Chapter III.

Background

In 1946, Gabor [23] introduced a method of representing signals simultaneously in time and frequency. A year later, he demonstrated this by using an optical/mechanical apparatus he designed and called the *frequency converter* [23]. The device sampled the speech signal—which was recorded on 35mm sound film—optically decomposed the signal into the Gabor spectrum and compressed the sampled signal onto a photocell via microscope lenses, and finally reconstructed the signal by refocusing the image on the photocell onto another sound film. Although much different in design, the frequency converter worked on a similar principle to that of the channel vocoder developed by Homer Dudley a few years earlier [17]. In essence, this principle determines a sliding interval of the signal and decomposes that portion of the signal into its constituent frequency components. One can think of the result of this analysis as a two dimensional representation of the signal with time on one axis and frequency on the other. The resolution in both the time and frequency axes is determined by the chosen interval.

This type of analysis differs from formal Fourier analysis, which would decompose the entire time signal into frequency components in a one-dimensional representation. Although one has access to all the frequencies of the signal using this method, one can not determine when in the signal these frequencies occur. Knowing where particular frequencies occur in a speech signal is crucial in speech processing.

The vocoder can be thought as computing the short-time Fourier transform [58:341–344]—also referred to as the complex spectrogram—therefore, it is not by accident that the Gabor transform and the short-time Fourier transform are related [6]. As the name implies the short-time Fourier transform determines the frequency components of short time samples of a signal. The most important distinction between the two is that the Gabor transform is more localized in frequency than the spectrogram, i.e., with the use of a Gaussian window function, the Gabor representation "provides the best spectral information for every point along the signal variation" [56:452].

Notwithstanding, Gabor's method did not become popular with the communication and signal analysis communities mainly owing to the lack of an explicit method of determining the Gabor coefficients—due to the non-orthogonality of the Gaussian window—and the frequency converter's impracticality for communication purposes [24]. Questions concerning the completeness and uniqueness of the Gabor expansion were also raised. The short-time Fourier transform, thus, became one of the main analytical tools in speech processing and continues to be today.

Interest in the Gabor representation waxed in the early 1980s as a result of two key findings. Bastiaans [5, 6] found a method of computing the Gabor coefficients by means of an auxiliary function. The importance of the finding rests with the fact that the coefficients can be determined using the efficient FFT [19]. Janssen [36], on the other hand, proved the completeness of Gabor expansions; however, he also proved that these expansions are not unique.

It is interesting to note that the Gabor representation has become popular in image analysis [7, 16, 56, 67]; however, it is still not very popular for analyzing temporal signals. Gabor originally introduced his frequency/time representation in order to study the compression of speech for communication purposes [24]—the main subject revisited in this research.

Gabor's contention that the Gaussian window provides the best resolution in time and frequency—first proven by Weyl [68]—has recently been questioned by [35, 61]. Weyl's proof is based on finding the product of the individual uncertainties in time and frequency. Wechsler and his colleagues [35, 61] argue that "...the proper way to measure conjoint resolution of a joint representation is to derive such a measure directly from the joint representation itself" [35]. Furthermore, these authors show that the Gabor representation and the short-time Fourier transform are smoothed versions of the (cross-)Wigner distribution [9, 10, 11].

Quite recently, the (cross-)Wigner, Gabor, and short-time Fourier transforms have been classified as special cases of Weyl-Heisenberg (W-H) frame integral operators [31, 32]. Duffin and Schaeffer [18] introduced frames for nonharmonic Fourier series expansions [15] in the early 1950s. In simple terms, a frame consists of a set of kernel functions or *wavelets*, which can completely characterize a given function by inner products of that function with the set of wavelets. Frames can be distinguished from bases in that neither orthogonality nor uniqueness is a requirement for frames. Frames are divided into two main classes: W-H frames and affine frames. As will be shown in the next section, the two frame classes are characterized by two distinct types of wavelets.

This background gave a very brief introduction to a variety of time/frequency analyses and their interrelationship. The short-time Fourier transform, the Gabor transform, and wavelet transforms in W-H and affine frames are more explicitly defined in the next few sections.

The Short-Time Fourier Transform

For convenience, all signals are limited to one-dimensional, continuous-time, square-integrable signals in this chapter. More specifically, $y, g \in L^2(\Re)$, where y

describes the signal function, g describes the window or wavelet function, and $L^2(\Re)$ is the Hilbert space over the reals. However these restrictions are not required [32]. In particular, the definitions and properties that follow are generalizable to two-dimensional spatial images and discrete signals as well. All proofs to the following properties may be found in the cited material.

As was mentioned in the previous section, the short-time Fourier transform finds the spectral characteristics of short segments of the signal $y(t)$. More specifically,

$$Y(\tau, \nu) = \int_{-\infty}^{\infty} y(t)\, g^*(t - \tau)\, \exp[-j2\pi\nu t]\, dt \tag{1}$$

where * denotes complex conjugation, and $t, \tau, \nu \in \Re$. As can be seen from Eq (1), the short-time Fourier transform is a function of two variables, τ and ν, associated with time and frequency respectively [6]. The function $g(t)$ is known as the window function. In wavelet parlance, $g(t)$ is the *mother wavelet*, and $Y(\tau, \nu)$ amounts to the inner product of the translates and modulates of $g(t)$ with $y(t)$ [32]. The original function $y(t)$ can be reconstructed from $Y(\tau, \nu)$ by

$$y(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} Y(\tau, \nu)\, g(t - \tau)\, \exp[j2\pi\nu t]\, d\tau\, d\nu \tag{2}$$

Eq (2) holds for any g. A simple proof of this was provided by Helstrom [33]. Eq (2) describes how the short-time Fourier transform expands a signal into a continuous set of translated and modulated wavelets or window functions. The short-time Fourier transform is also known for discrete signals and has been used for years in speech and signal processing [58]. Notwithstanding, the signal processing perspective of the short-time Fourier transform is somewhat different from the perspective just provided. The distinction is important; however, it will not be dealt with until Chapter III.

The Gabor Representation

In contrast to the short-time Fourier transform, the Gabor representation expands a signal $y(t)$ into a *discrete* set of translated and modulated *elementary* [23] or window functions $G_{m,k}(t)$ [6]. $G_{m,k}(t)$ is defined as follows:

$$G_{m,k}(t) = g(t - m\alpha)\ \exp[j2\pi k\beta t] \tag{3}$$

where $k, m \in \mathcal{Z}$. The parameters α and β are associated with the uncertainties Δt and Δf, respectively, found when attempting to simultaneously define a signal in both time and frequency [23:432]. The uncertainty relationship between Δt and Δf is defined as

$$\Delta t \Delta f \geq 1/2 \tag{4}$$

This relationship is known as the Heisenberg uncertainty principle in the quantum mechanics analogue. Weyl found that a Gaussian window $g(t)$ (see Eq (10)) minimized Eq (4) [23, 68].

With this in mind, the Gabor expansion of $y(t)$ is defined as

$$\begin{aligned} y(t) &= \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} C_{m,k}\, G_{m,k}(t) \\ &= \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} C_{m,k}\, g(t - m\alpha)\ \exp[j2\pi k\beta t] \end{aligned} \tag{5}$$

As stated earlier, because of the non-orthogonality of the Gaussian g, solutions to the Gabor coefficients were not found until recently. It so happens that an infinite number of solutions to $C_{m,k}$ are possible [36]. Bastiaans [5, 6] found the following solution:

$$C_{m,k} = \int_{-\infty}^{\infty} y(t)\, \gamma^{*}(t - m\alpha)\ \exp[-j2\pi k\beta t]\, dt \tag{6}$$

The function $\gamma(t)$ is known as the biorthogonal function of $g(t)$ and is defined by the relationship

$$\int_{-\infty}^{\infty} g(t)\,\gamma^{*}(t - m\alpha)\,\exp[-j2\pi k\beta t]dt = \delta_k\,\delta_m \tag{7}$$

where $\delta_n = 1$ for $n = 0$, and $\delta_n = 0$ for $n \neq 0$ (the Kronecker delta)[6, 19, 22]. The biorthogonal function $\gamma(t)$, in turn, can be found via the Zak transform [22, 37]. The Zak transform of $g(t)$ is defined as

$$(Zg)(\tau,\omega) = \sum_{k=-\infty}^{\infty} T^{1/2}\,g(\tau + kT)\,\exp[-j2\pi k\omega T],$$
$$0 \leq \tau \leq T,\ 0 \leq \omega \leq T^{-1} \tag{8}$$

where T is the sampling interval. One can now obtain $\gamma(t)$ by

$$\gamma(\tau) = \int_0^1 \frac{d\omega}{(Zg)^{*}(\tau,\omega)} \tag{9}$$

For Gabor's original Gaussian window

$$g_G(t) = \left(\frac{\sqrt{2}}{T}\right)^{1/2} \exp\left[-\pi\,(t/T)^2\right] \tag{10}$$

Bastiaans [5, 6] found its biorthogonal function to be

$$\gamma_G(t) = \left(\frac{1}{T\sqrt{2}}\right)^{1/2} (K_0/\pi)^{-3/2} \exp\left[\pi\,(t/T)^2\right] \sum_{n+1/2\geq t/T} (-1)^n \exp[-\pi(n+1/2)^2] \tag{11}$$

where $K_0 = 1.85407468$. Eqs (10), and (11), with $T = 1$, are plotted in Figure 1.

Others have investigated alternative window functions for the Gabor representation [6, 22]. Of special interest is Friedlander and Porat's [22] investigation of the one-sided exponential window $g_E(t)$ for representing transient signals. The function $g_E(t)$ and its

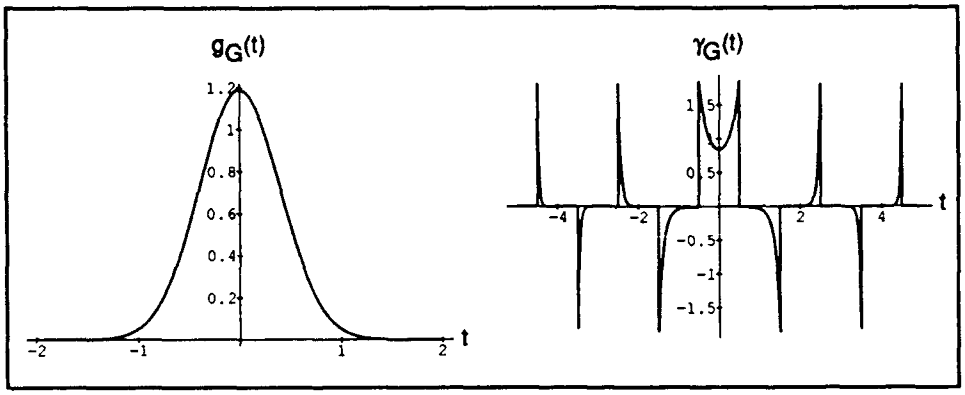

Figure 1. The Gaussian Window (left) and its Biorthogonal Function (right)

biorthogonal function $\gamma_E(t)$ are defined as follows:

$$g_E(t) = \exp[-t/\tau]\, u(t) \tag{12}$$

$$\gamma_E(t) = \exp[t/\tau]\, [-u(t+1) + 2u(t) - u(t-1)] \tag{13}$$

where $u(t)$ is the unit step function and τ is the time constant. The plots of Eqs (12), and (13) are illustrated in Figure 2.

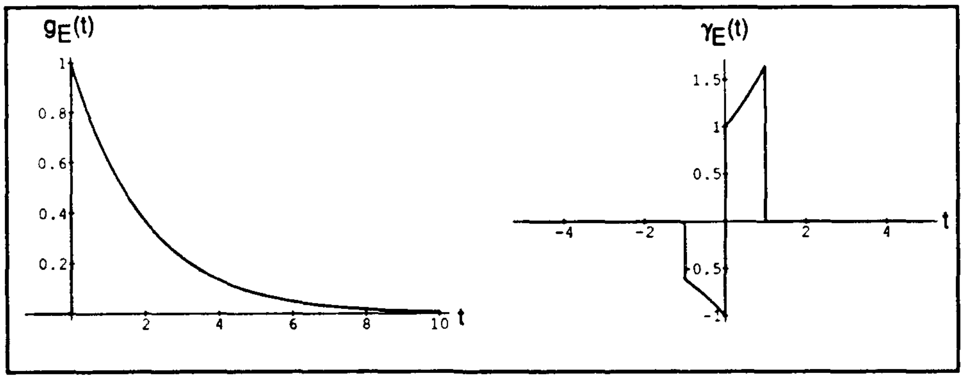

Figure 2. The Exponential Window (left) and its Biorthogonal Function (right)

Janssen [37:26] found the following alternative method of determining the Gabor coefficients using the Zak transform of the signal and window functions:

$$C_{m,k} = \int_0^1 \int_0^1 (Zy)(\tau,\omega)\,(Zg)^{-1}(\tau,\omega)\,\exp[j2\pi m\tau t - j2\pi k\omega t]\,d\tau\,d\omega \qquad (14)$$

The main advantage of Eq (6) over Eq (14) is that Eq (6) can be conveniently and easily approximated via the FFT [19, 22](see Eqs (39), (40), and (41)).

The two-dimensional Gabor representation is illustrated in Figure 3. As the figure shows, the magnitude of the Gabor coefficients $|C_{m,k}|$ are shown uniformly spaced in time and frequency as defined by $\alpha = \Delta t$ and $\beta = \Delta f$.

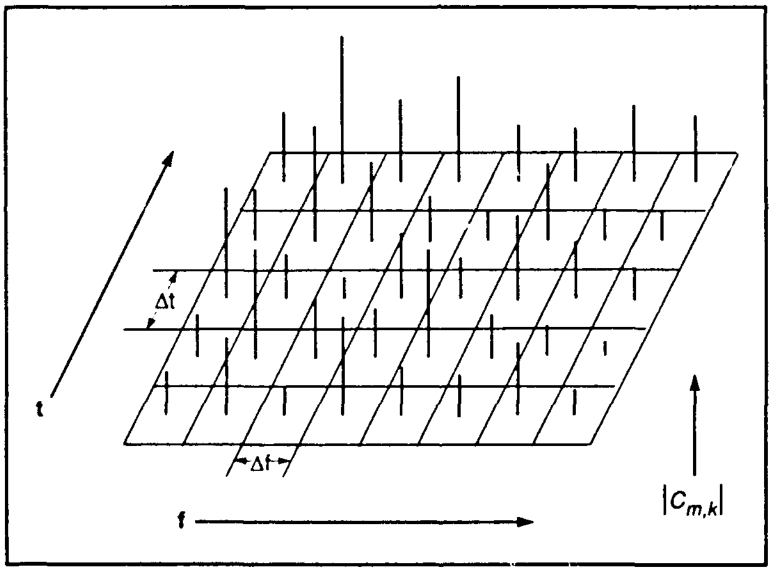

Figure 3. The Gabor Time/Frequency Analysis

Wavelets

Introduction In this section some very basic definitions of wavelets and their associated frames are given. Up-to-date surveys of wavelets and their associated frames have been written by Heil and Walnut [31, 32]. Reference [31] is geared towards the engineer. The main contributions to the field of wavelets include the works of I. Daubechies, A. Grossman, Y. Meyer, and J. Morlet [15, 13, 14, 28]. Other introductions to wavelets can be found in [12, 66].

As was discussed at the outset of this chapter, the short-time Fourier, Gabor, and (cross-)Wigner distributions are examples of W-H frame operators [32]. These transforms compute the inner product of a given function $y(t)$ with the translates and modulates of a function—or wavelet—$g(t)$. Affine frame operators are characterized by transforms that also compute the inner product of a function with a wavelet; however, in contrast to W-H frames, these wavelets are translated and dilated—or constricted, depending on how the analysis is defined. This is shown in the following subsection.

One of the developments in frame research that is of particular importance to signal processing is that a method for finding wavelets that produce orthonormal bases in affine frames has been found [13]. Orthogonality is important when reconstruction from a compressed set of frequency components is required. The issue of orthogonality will be revisited in greater depth in the next chapter.

Affine Wavelet Transform The affine wavelet transform is defined in this subsection. To differentiate between the two types of wavelets and to use the notation found in the wavelet literature, let $\psi(t)$ correspond to a wavelet function in an affine frame. The affine wavelet transform is then defined as [32]

$$(Wy)(a, \epsilon^b) = \int_{-\infty}^{\infty} y(t)\, \epsilon^{b/2} \psi\left(\epsilon^b t - a\right)\, dt \tag{15}$$

where $a, b, \epsilon \in \Re$. The function $y(t)$ can be reconstructed from $(Wy)(a, \epsilon^b)$ by

$$y(t) = \int_0^{\infty} \int_{-\infty}^{\infty} (Wy)(a, \epsilon^b)\, \epsilon^{b/2} \psi\left(\epsilon^b t - a\right)\, da\, db \tag{16}$$

As in the case of the other transforms discussed in the preceding two sections, the discrete versions of Eqs (15), and (16) exist [12].

A specific case of a wavelet is now defined and used to explain the consequence of Eq (15) [12]. Consider an orthonormal basis described by translating and constricting the mother wavelet $\psi(t)$, where $\psi(t)$ is defined by

$$\psi(t) = \begin{cases} 1, & 0 \leq t < 1/2 \\ -1, & 1/2 \leq t < 1 \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

where $\psi_l^k = 2^{l/2}\psi(2^l t - k)$; $l, k \in \mathcal{N}$; and the support of ψ_l^k is the interval, I, defined by the inequality $0 \leq 2^l t - k < 1$. This orthonormal basis ψ_l^k is known as the Haar basis [13]. Eq (17) is plotted in Figure 4.

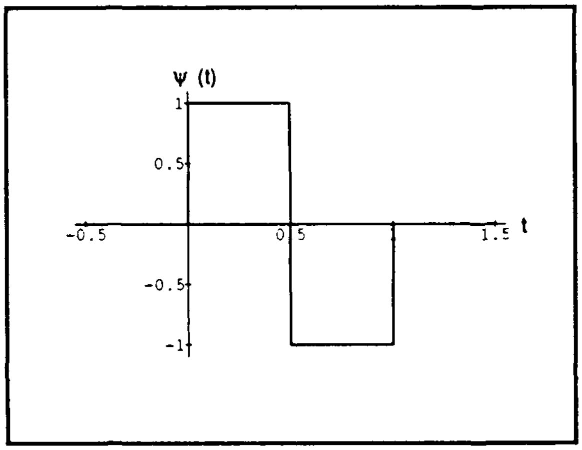

Figure 4. The Haar Mother Wavelet

Eq (15), with ψ_l^k in the integrand, calculates the frequencies proportional to 2^l contained in y about the interval I. In a linear filtering perspective, Eq (15) can be modeled by correlating y with a series of overlapping bandpass filters that increase in bandwidth logarithmically—base 2 in this example–as a function of the filter's center frequency.

An alternative explanation can be defined in terms of a multilevel resolution analysis. Let l denote the level of analysis. Furthermore, for simplicity, let the analysis be performed over the interval I, defined by the inequality $0 \leq t < 1$, of the signal $y(t)$. At the first level, $l = 0$, the analysis of $y(t)$ is performed with the mother wavelet $\psi_0^0 = \psi(t)$. The analysis amounts to the inner product of the two functions. At the next level of resolution, $l = 1$, the wavelet is constricted or shrunk along the time axis by $1/2$, and an inner product is again computed. Since the interval covered by the wavelet at this resolution is $I/2$, the wavelet is translated once ($k = 1$) and a second inner product is computed in order to cover the entire interval I. The wavelet at this level becomes $\psi_1^k = 2^{1/2}\psi(2t - k)$, where $k = 0, 1$. The analysis at subsequent levels is performed similarly by constricting the mother wavelet by 2^{-l} at level l and computing the inner product of $y(t)$ and ψ_l^k for $k = 0, 1, \ldots, 2^l - 1$. The Haar wavelets at levels 0–3 and at all translations to cover I are shown in Figure 5.

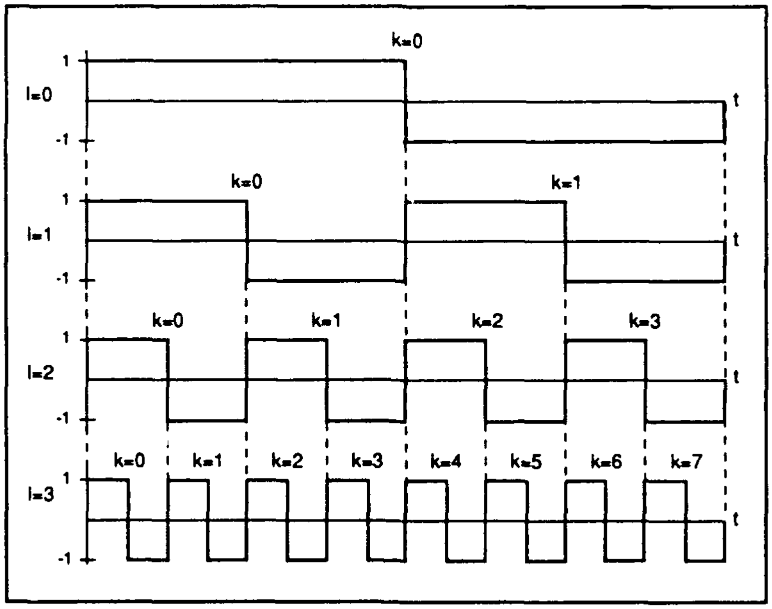

Figure 5. The Haar Wavelet at Four Resolutions

The Fourier transform of ψ_l^k has a bandwidth proportional to 2^l. This means the bandwidth is doubled at each subsequent level of resolution. The time/frequency analysis performed by ψ_l^k is illustrated in Figure 6. The vertical bars correspond to the amplitude of the correlation coefficients that result from the wavelet analysis just described. The value of the coefficients are defined by their height, which is arbitrary in this example for instructive purposes. The coefficients are also shown as positive values to simplify the drawing. The frequency axis is labeled f, and the time axis is labeled t. Figure 6 shows that the interval of the analysis performed by a wavelet at each subsequent level decreases by one half and its bandwidth doubles, as expected from the previous discussion.

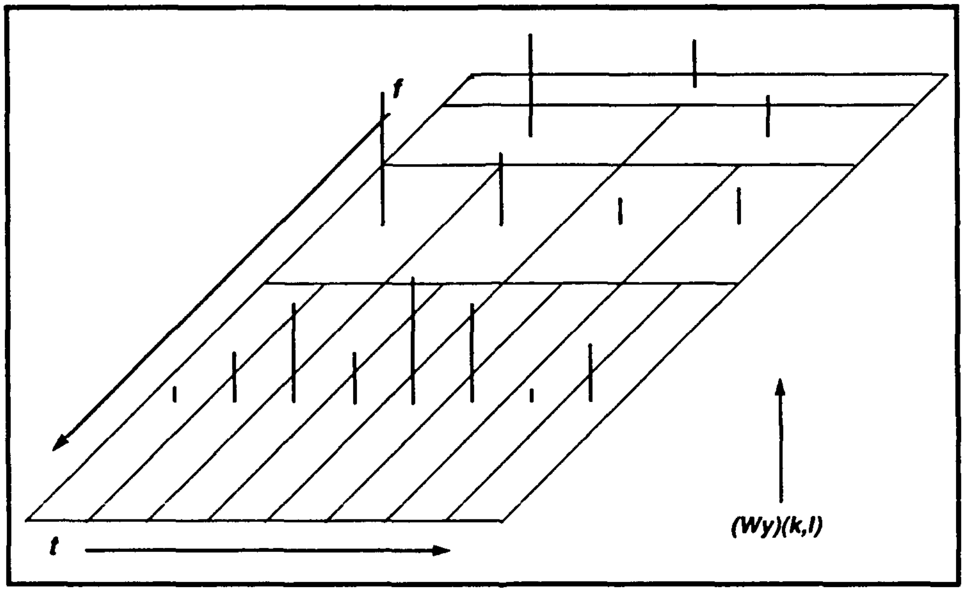

Figure 6. The Affine Wavelet Time/Frequency Analysis

Meyer and others have defined an entire class of smooth orthonormal bases of the form ψ_k^l [13], which perform $\log_2$

analyses of signals. However, the logarithmic analysis of the wavelet transform does not have to be in base 2. The affine wavelet transform analyzes functions in a logarithmic scale of base b in general. This characteristic is what differentiates affine wavelet analysis from W-H frame analysis. The W-H wavelet analysis localizes *uniformly* in time and frequency, whereas the affine wavelet analysis localizes *logarithmically* in time and frequency. This difference can be seen by comparing the illustrations of the Gabor time/frequency analysis and the affine wavelet time/frequency analysis in Figures

3 and 6. Thus, the affine transform may have advantages over W-H transforms for speech processing and other nonstationary signal processing, because it analyses signals over multiple scales of resolution. In contrast, for example, the Gabor analysis requires that a specific Δt and Δf be defined for a single resolution analysis. It may not always be easy to find these values.

The analysis of signals in a logarithmic frequency scale performed by the affine wavelet transform is also performed, to a great extent, by the inner ear. In the following section, the parallels between the dynamics of the ear and wavelet frames are explored.

Biological Considerations

Introduction This section provides a very brief introduction to the anatomy and physiology of hearing. The discussion concentrates around the topics of most relevance to this research. For a comprehensive, current introduction to the science of hearing see Gulick, et al [29]. Most of the material covered in the next subsection is a summary of portions of that text.

The Auditory Periphery The peripheral auditory system consists of the external ear, the middle ear, and the internal ear. This system acts as a transducer, which converts air sound pressure into electrical impulses that our brains can process. In addition, the peripheral auditory system acts as a system of bandpass filters. These filters define the frequency range that animals can perceive. In humans, this frequency range is approximately 20 Hz–20 kHz [29:217].

The external ear is composed of the *pinna*, the *external meatus*, and the *tympanic membrane*. The pinna and the meatus—commonly referred to as the ear and ear canal respectively—act as a first stage filter that increases the amplitude of frequency components in the range of approximately 500 Hz–10 kHz [29:87]. At the end of the meatus, lies the flexible tympanic membrane, or ear drum, which is set into complex patterns of vibratory motion by external sound waves. The tympanic membrane forms the boundary between the external ear and middle ear. Thus, to summarize the function of the external ear, sound waves are collected, filtered, and propagated to the tympanic membrane by the pinna and the meatus. The tympanic membrane then responds to these sound waves by vibrating.

Beyond the tympanic membrane lies the middle ear. The middle ear is comprised of a cavity containing a chain of three small bones: the *malleus*, *incus*, and *stapes*. This chain connects the tympanic membrane to the oval window of the *cochlea*. The fluid filled cochlea is the main sensory organ of hearing and comprises the internal ear. The function of the bony chain, therefore, is to transmit the vibratory action of the tympanic membrane to the sensory receptors in the cochlea, and to provide an impedance match between free space and the liquid filled cochlea.

The cochlea—Latin for snail—is a coiled cavity in the temporal bone containing three fluid filled chambers: the *scala vestibuli*, *scala media*, and *scala tympani*. A stiff membrane, called the *basilar membrane* separates the scala media and scala tympani. In humans, the basilar membrane is approximately 35mm in length, .5mm wide at the apex of the cochlea, and .08mm wide at the base (towards the oval window). The basilar membrane has a logarithmic gradient of stiffness that increases along its length from the apex to the base of the cochlea. Both the tapered shape of the basilar membrane in conjunction with the stiffness gradient produce the logarithmic filtering characteristics mentioned earlier. A collection of structures called the *organ of Corti* rests on the basilar membrane. The most important of these structures are the *tectorial membrane*, the *inner hair cells*, and the *outer hair cells*. The hair cells are organized along the basilar membrane in rows, with the tectorial membrane covering these cells like a canopy.

The cochlear dynamics are now described. As the stapes moves back and forth at the oval window, longitudinal waves are produced in the fluid of the interior chambers of the cochlea. These waves produce localized oscillations of the basilar membrane whose positions are a function of the external frequency of the sound. For example, a 1 kHz tone produces a maximum pattern of displacement at a precise location along the length of the basilar membrane. The action of the basilar membrane produces shearing forces on the inner hair cells via the tectorial membrane. When this occurs, the inner hair cells produce electrical signals that are transmitted to subsequent processing layers of the brain along the cochlear nerve. Therefore, the cochlea would signal the occurrence of a pure 1 kHz input by breaking into oscillations at a specific *place* along its length.

Complex sounds composed of a number of frequencies are encoded along the basilar membrane by oscillatory displacements at specific locations corresponding to the frequencies of the sound. In other words, the basilar membrane can be thought as a spatial frequency template or *tonotopic* map. In addition to this spatial information, temporal information is also produced by the firing rates of the inner hair cells. The preferred or *characteristic frequency* (CF) of an inner hair cell is the frequency that produces the maximal displacement of the basilar membrane where that hair cell resides. Hair cells with CFs of 4 kHz and below fire at the same rate as their CF. This phenomenon is known as *phase synchrony*. It is not known how the brain uses this spatio/temporal information; however, it is known that tonotopicity is maintained at all the relay stations leading to the neocortex, as well as at the primary sensory fields of the auditory cortex. The assumption, thus, is that spatial patterns are important to the brain. The meaning of the temporal information is more difficult to decipher.

The spatial encoding characteristics of the basilar membrane should not be construed as only a simple Fourier analysis capability. The situation is much more complex than that and still largely not understood. This is illustrated by a phenomenon called *masking* [29:300–313]. As was mentioned earlier, a pure tone produces localized patterns of electrical activity in fibers in the auditory nerve whose CFs coincide with the tone's frequency. It has been found that the firing rate of an auditory nerve fiber can be suppressed or masked by presenting tones of similar frequencies [62]. Furthermore, masking is not possible after the masking tones have deviated from the CF by a certain amount. Therefore, masking occurs within certain frequency bands called *critical bands* [29:306]. How these bandwidths are determined is a subject of controversy [21, 50, 70]; however, the general consensus is that the masking phenomenon suggests that the cochlea acts as system of overlapping bandpass filters. Although there is a question as to exactly how wide these filters are, it is known that their bandwidths increase logarithmically as a function of the CF. Patterson [54] has determined the shape of these filters using masking experiments and found them to be cone shaped. Figure 7 illustrates this phenomenon. The intensity and bandwidth of the filters shown in Figure 7 are not precise; they are meant to illustrate the filter function of the cochlea.

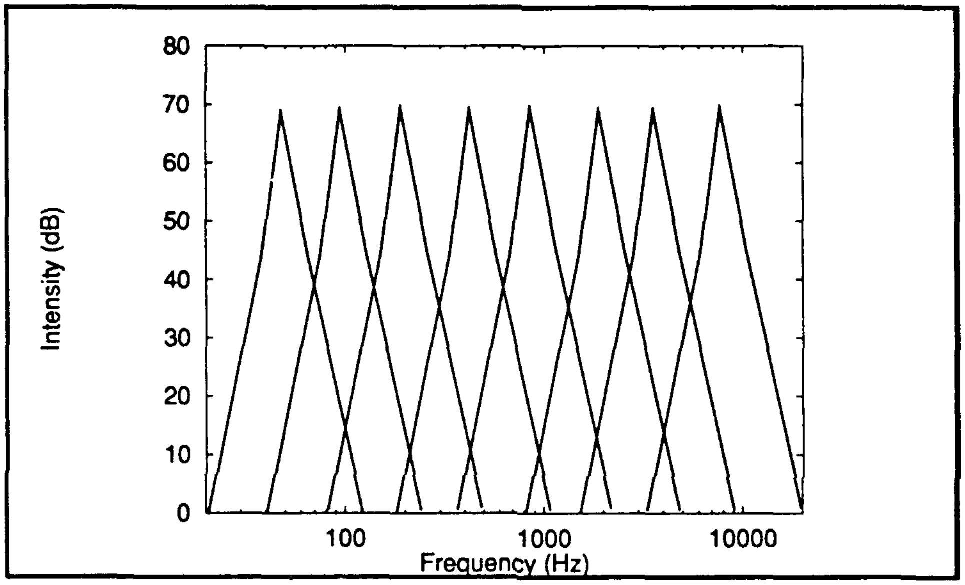

Figure 7. Filter Model of the Cochlea; Adapted from [29:307]

The simple description of the anatomy and physiology of the cochlea just given does little justice to the complexity of this system. Much is still not known about the function of the cochlea, in particular the feedback system, called the *efferent* system, which incorporates the outer hair cells.

As is evident from the previous section, the computational analysis performed by the ear is reminiscent of the analysis performed by the affine wavelet transform. The subsequent processing of information in the numerous auditory nuclei beyond the cochlear nerve is not understood. However, evidence for one type of processing has been accumulating over the last 15 years. This type of processing is called lateral inhibition, or is also known as on-center off-surround processing. LINs are examined next subsection.

Lateral Inhibition Networks Lateral inhibition networks were first found by Hartline and Ratliff [59, 60] in the eye of the horseshoe crab (*Limulus*). Since Ratliff's discovery, lateral inhibition has been found to be a common processing strategy in all sensory systems across species [64], including the auditory system [40, 63]. LINs perform several functions; the most important being contrast enhancement. In contrast enhancement, spatial peaks and edges as well as temporal changes are highlighted. Other important functions of LINs include noise suppression, automatic gain control, buffering of information for short-term memory, and a variety of oscillatory functions. These effects have been modeled and studied by many over the years [20, 25, 27, 59, 60, 63].

As discussed above, many LIN models have been developed (see [26] for a review). The one adopted in this research is based on the non-linear, shunting cell membrane equation (see Eq (18)) found by Hodgkin and Huxley [26, 34] in electrophysical studies of the giant squid axon. In a study that investigated the effects of lateral inhibition on the output of a cochlea model, Shamma [63] used an additive version of the shunting equation in his LIN model. Elias and Grossberg [20], however, have shown that the shunting model is much less susceptible to oscillatory behavior than the additive model, and therefore justifies the use of the shunting model in this research. Before describing the membrane equation, a structural description of a typical LIN will be presented.

Figure 8 illustrates the cell interactions of a *recurrent* LIN. The cells or *nodes* of the system are labeled v_n. Each node has an input g_n and produces an output $f(x_n)$. The parameter x_n is called the activity of node v_n. The arrows illustrate excitatory connections or positive gain communication lines, and the small circles illustrate inhibitory or negative gain connections. To simplify the illustration, only the connections exiting node v_i are shown. The outputs of all other nodes are considered to be similar to the outputs of node v_i.

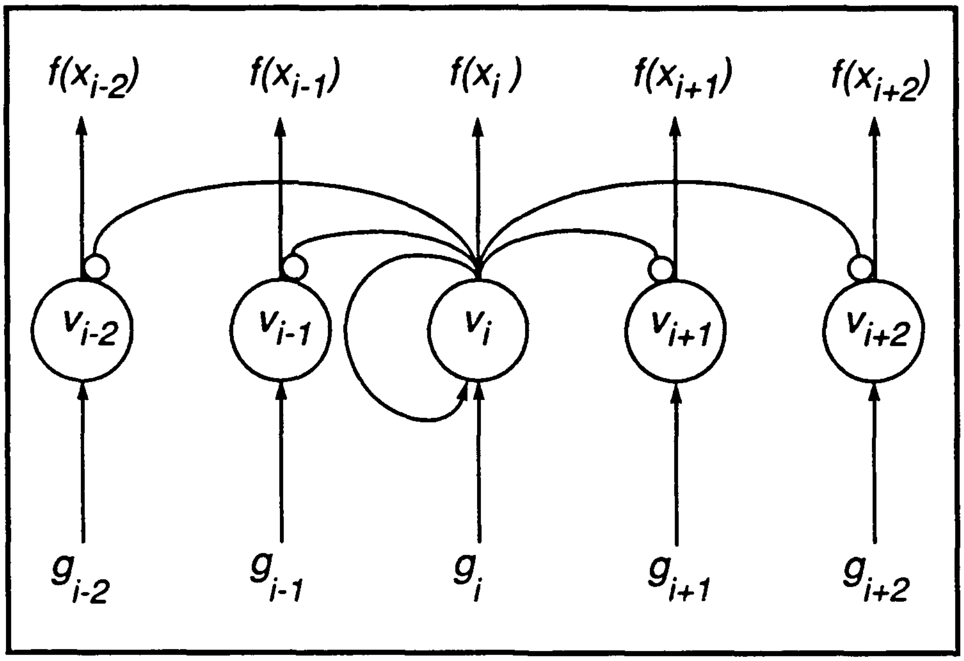

Figure 8. Lateral Inhibition Network Model

As Figure 8 shows, LIN interaction can described as a process cell self excitation and neighborhood inhibition. The pattern of excitation does not have to be limited to individual nodes. Groups of nodes can mutually excite one another and in turn inhibit a surrounding neighborhood. The networks are also not restricted to one dimension. In fact, two-dimensional LINs are the ones found in the cortex.

The activity and output equations for each node can now be defined as in Eq (18) and in Eq (19) respectively. Although the x_i and g_i are functions of time, the variable t has been omitted in the following equations for simplicity.

$$\frac{d}{dt}x_i = -x_i + (A - x_i)\left[g_i + f(x_i)\right] - (B + x_i)D\sum_{j\neq i} f(x_j) \tag{18}$$

$$f(x_i) = \begin{cases} x_i^2/(x_i^2 + \alpha^2), & x_i \geq 0 \\ 0, & x_i < 0 \end{cases} \tag{19}$$

All properties of Eq (18) can be found in [20, 25, 27]. Eq (19) defines a sigmoidal function. Other sigmoid functions have been used, in particular $f(x) = 1/(1 + \exp[-x])$.

The shunting Eq (18) is in the form of Eq (20) found by Hodgkin and Huxley [34], which describes the membrane potential V of neurons .

$$C\frac{d}{dt}V = (V^p - V)g^p + (V^+ - V)g^+ + (V^- - V)g^- \qquad (20)$$

C relates to a constant lipid membrane capacitance. The constants V^p, V^+, and V^- relate to the passive, excitatory, and inhibitory membrane potential saturation points respectively. In the Hodgkin and Huxley model, these saturation points indicate the equilibrium potential of specific ions. The terms g^p, g^+, and g^- are the passive, excitatory, and inhibitory ionic conductances respectively. By simple changes of variables, Eq (18) and Eq (20) can be related. Capacitance C becomes constant one in the shunting model. Membrane potential V is characterized by activity x_i. Membrane potential saturation points V^p, V^+, and V^- become constants 0, A, and B respectively. Likewise, the passive ionic conductance g^p equals one, the excitatory conductance g^+ equals the input to the cell g_i plus feedback $f(x_i)$, and the inhibitory conductance g^- equals an inhibitory gain D times the sum of the outputs of the competing neurons $\sum_{j \neq i} f(x_j)$. In reality, there are many more ionic conductances than those shown in Eq (20) (see [64]), but these can be grouped functionally into the three main types of conductances: passive conductances, that bring the cell back to its resting potential; excitatory conductances, that raise the cell membrane potential towards the signaling potential; and inhibitory conductances, that suppress the cell membrane potential and cell firing.

How do LINs relate to what has been described thus far? One of the main problems addressed in this research is finding compression codes for speech. LINs have properties which make them potentially useful for eliminating redundant frequency or correlation terms in wavelet—W-H or affine—transform space. This will be addressed in depth next chapter.

Summary and Final Comments

In this chapter, wavelets and wavelet frames were introduced. To recap, wavelet frames are divided into two classes: Weyl-Heisenberg frames and affine frames. Most time/frequency distributions, including the short-time Fourier transform and the Gabor transform, belong in the Weyl-Heisenberg frame class. These distributions are characterized by transforms that compute the inner product of a function with translates and modulates of a wavelet. The affine wavelet transforms, on the other hand, are characterized by transforms that compute the inner product of a function with translates and dilates of a wavelet. It was also shown that the affine wavelet transform can be thought as computing a correlation between a signal and a series of overlapping bandpass filters whose bandwidths increase logarithmically as a function of frequency. Evidence was then presented that suggests that the peripheral auditory system, particularly the cochlea, performs a similar analysis to that of the affine wavelet transform. The final section dealt with the architecture and dynamics of LINs.

Next chapter consists of the system design for decomposing, compressing, and reconstructing speech. The full connection between LINs and wavelet transforms is shown.

III. System Design

Introduction

This chapter details the methods used to decompose, compress, and regenerate speech signals in this research effort. Three transforms are used for this purpose: the short-time discrete Fourier transform, the discrete Gabor transform, and the discrete affine wavelet transform. In the first section, the software and signal processing environments are described. Several definitions and the notation used throughout the remainder of this document is provided immediately after. The remainder of this chapter concentrates on defining the discrete versions of the transforms introduced in the last chapter, and on the subsequent interaction of LINs. A variety of window or wavelet functions are explored for each transform defined. Compression is achieved by eliminating *local* low energy correlation or frequency components produced by the varied transforms via the dynamics of LINs. The interaction between these networks and the transform outputs is fully discussed.

System Environment

As was mentioned in the introductory chapter, all digital signal processing and software development is performed on the NeXT computer. The speech signals are sampled at 8012.8 Hz by the CODEC and quantized using 8-bit μ-law logarithmic compression [65:76–78]. Each sampled and quantized speech signal is then stored in Objective C files by resident software routines. All further processing of the speech data is performed by software written in a combination of Objective C, ANSI C, and C macros wrapped around assembly language vector processing routines for the resident Motorola 56001 Digital Signal Processor—hereafter called the DSP. Most array processing, including FFTs, is performed by the DSP. Whatever processing is not performed by the DSP is performed by the NeXT's Motorola 68030 in floating point format.

The data in the speech files are easily accessible using Objective C routines. These routines convert the data from 8-bit μ-law to a 16-bit linear format. The speech data are again converted—this time to floating point—so that the data can be scaled between values small enough to prevent overflow errors in the DSP. All values in the DSP must be maintained between -1 and 1 in a 24-bit linear format; thus, the floating point values are scaled then converted to 24-bit linear. The linear scaling function $S(x)$ used is described as

$$S(x) = \frac{x - m_o}{x_o - m_o}(x_n - m_n) + m_n \tag{21}$$

where

x_o = the maximum value of the range of x

m_o = the minimum value of the range of x

x_n = the maximum value of the new range

m_n = the minimum value of the new range

Outputting the processed speech is nearly a reversal of the process just explained. The 24-bit DSP data are converted to floating point and processed further if need be. The completely processed data are finally converted back to 16-bit linear and stored in Objective C sound files. Resident software using mouse activated windows can now be used to activate the CODEC and play back the synthesized sound files.

Definitions and Notation

The discussion and definitions in Chapter II dealt with the decomposition and reconstruction of continuous time signals $y(t)$. In this and subsequent chapters, operations will be performed on discrete sequences $y(n)$, where $n \in \mathcal{Z}$. Discrete $y(n)$ is obtained from the continuous time signal by sampling $y(t)$ at rate $1/T$. More precisely,

$$y(n) = y(t)|_{t=nT} = y(nT) \tag{22}$$

Time dependence is dropped by omitting the sampling period T in Eq (22).

Where possible, the discrete Fourier transform (DFT)—by means of the FFT—will be used to decompose signals. The DFT and inverse DFT are defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp[-j2\pi kn/N], \ 0 \leq k \leq N-1 \tag{23}$$

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) \exp[j2\pi kn/N], \ 0 \leq n \leq N-1 \tag{24}$$

respectively, where N is the number of samples in $x(n)$ and $1/N$ is the fundamental digital frequency in cycles. Therefore, the frequency index k is related to the digital and analog frequencies f_k and ν_k, respectively, as follows [43:287]:

$$\underset{\text{Frequency Index}}{k} \longrightarrow \underset{\substack{\text{Digital Frequency}\\ \text{(cycles)}}}{f_k = k/N} \longrightarrow \underset{\substack{\text{Analog Frequency}\\ (Hz)}}{\nu_k = k/NT} \tag{25}$$

As previously mentioned, the sampling rate for all signals is $1/T = 8012.8$ Hz. All DFT sequences are of length $N = 256$, which relates to a window of $NT = 31.95$ msec. Thus, the fundamental analog frequency is $f_1 = 1/NT = 31.3$ Hz. A 31.95 msec. window length is reasonable since it is known that speech is relatively stationary (i.e., the statistics of the signal remain constant) in periods of approximately 30 msec. or less.

The windowed versions of Eq (23) and Eq (24) are used for short-time Fourier and Gabor decomposition. The next two sections detail both the decomposition and synthesis of speech signals using these methods.

Short-Time Fourier Decomposition/Reconstruction

Introduction In the previous chapter, the short-time Fourier transform was introduced and described in the perspective of the more general theory of frames; in particular, Weyl-Heisenberg frames. In that perspective, decomposition of signals amounts to a projection onto a basis defined by $g(t-\tau)\exp[j2\pi\nu t]$, or the translates and modulates of $g(t)$. The bases defined in this manner need not be orthogonal to form complete sets. However, for discrete translations and modulations of g—as in the case of the Gabor expansion—care must be taken when defining g. The non-orthogonality of the Gaussian g created difficulties in finding a solution to the Gabor coefficients until Bastiaans and Janssen found the solutions defined earlier.

The traditional perspective taken by the signal processing community is somewhat different. As in the perspective of frames, successive intervals of the desired signal to be decomposed are analyzed one at a time. However, the analysis of the signal is always considered to be an orthogonal expansion into a combination of basis functions defined by the complex exponent, $\exp[j2\pi\nu t]$. In other words, the window function g is not considered part of the basis but part of the signal to be analyzed. The choice of g depends on the ease by which the original, or close approximation of the function y can be resynthesized from the windowed expansion, and on the specific frequency characteristics of g itself (see Harris [30] for a study on windows and their characteristics). Thus, the question of whether the function g produces an orthogonal set or not rarely comes up. An in-depth treatment of discrete short-time Fourier analysis is found in [58:250–354].

The Short-Time Discrete Fourier Transform Three window functions are used in the short-time Fourier decomposition of speech signals in this research. These are the rectangular window, the Hamming window, and the compactly supported Gaussian window. These windows along with the short-time versions of Eqs (23) and (24) are defined in this section. In addition, a modification to the inverse short-time DFT is defined, which is necessary for reconstructing good quality speech from a compressed Fourier space.

The rectangular, Hamming, and compactly supported Gaussian windows are defined as

Rectangular:

$$g_R(n) = \begin{cases} 1, & 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

Hamming:

$$g_H(n) = \begin{cases} 0.54 - 0.46\cos[2\pi n/(N-1)], & 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

Gaussian:

$$g_{\tilde{G}}(n) = \begin{cases} \exp[-4(\frac{n-N/2}{N})^2], & 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

The plots of each of these windows are shown in Figures 9, 10, and 11 respectively.

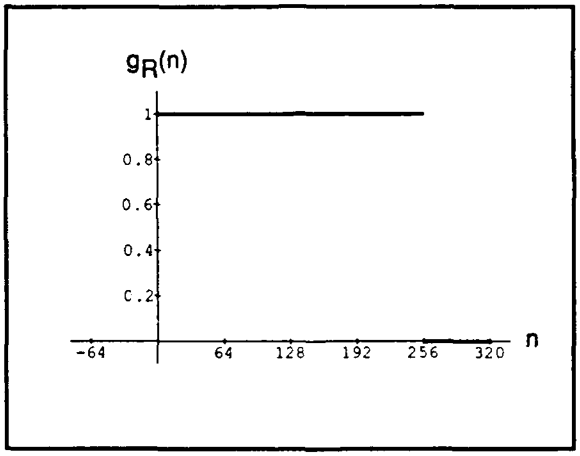

Figure 9. The Rectangular Window

The decomposition and reconstruction formulas for $y(n)$ depend on the window used. This is shown by first defining the general form of the short-time DFT pair as

$$Y(m,k) = \sum_{n=0}^{N-1} y(n)\, g(n-mN)\, \exp[-j2\pi kn/N],\ 0 \leq k \leq N-1 \tag{29}$$

$$y(n)\, g(n-mN) = 1/N \sum_{k=0}^{N-1} Y(m,k)\, \exp[j2\pi kn/N],\ 0 \leq n \leq N-1 \tag{30}$$

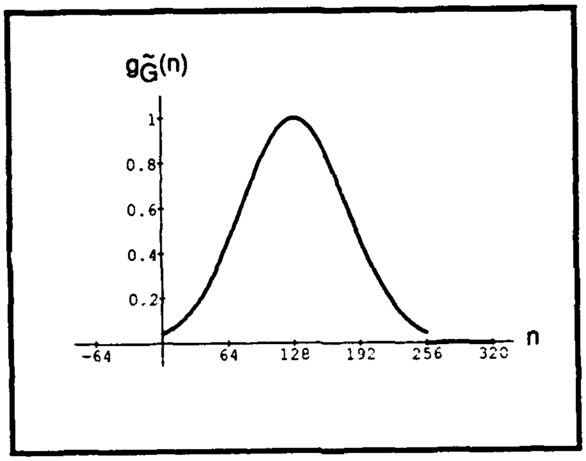

Figure 10. The Compactly Supported Gaussian Window

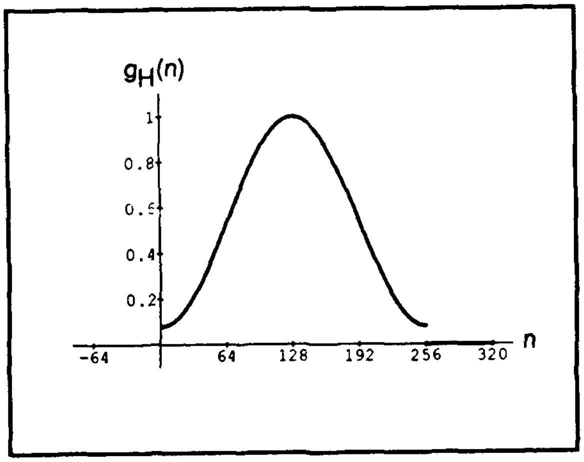

Figure 11. The Hamming Window

where $m \in \mathcal{N}$. When a rectangular window is used, $y(n)$ can be decomposed and exactly recovered by Eq (30). Since $g_R(n - mN) = 1$ by definition, $y(n)$ is recovered exactly by summing over all window shifts. More precisely

$$\sum_{m=0}^{M-1} y(n)\, g_R(n - mN) = y(n) \tag{31}$$

where M represents the number of subsequences of length N that describe the finite sequence $y(n)$. Meaning that, $y(n)$ is of length MN. The decomposition and reconstruction formulas for $y(n)$ with a rectangular window thus becomes

$$\begin{aligned} Y_R(m,k) &= \sum_{n=0}^{N-1} y(n)\, g_R(n - mN) \exp[-j2\pi kn/N],\ 0 \le k \le N-1 & (32) \\ y(n) &= 1/N \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} Y_R(m,k) \exp[j2\pi kn/N],\ 0 \le n \le N-1 & (33) \end{aligned}$$

If $g(n)$ is not rectangular then the right side of Eq (33) will not result in $y(n)$ but in $y(n)g(n - mN), \forall m$. In other words, the result is $y(n)$ multiplied by all translations of $g(n)$. Figure 12 shows this result. The top graph shows the original $y(n)$, a sampled sinusoid. Directly underneath, the consequence of Eq (33), in the case g is Gaussian, is shown.

Nevertheless, an approximation of $y(n)$ can still be reproduced if the windows are overlapped. That is, if $g(n)$ is repeatedly translated by $R < N$, then $y(n)$ can be approximated. To illustrate this, Figure 13 shows the plot of $S_g(n) = \sum_r g_{\tilde{G}}(n - rR)$, for $R/N = 0.50$, or 50% overlap. The ideal result of summing the overlapped window function would be the unit function, as illustrated by the gray line. In this ideal case, y could be exactly recovered. However, it can be seen that for the Gaussian window a low amplitude *wobble*—with a fundamental frequency of 62.6 Hz in the present case–results.

The error in $y(n)$ can be reduced with more overlap of $g(n)$. Figure 14 shows this for $S_g(n)/2$ with $R/N = 0.25$. The plot is scaled to show the small peak error in this case, which is approximately 15×10^{-3}. A similar situation arises when the Hamming

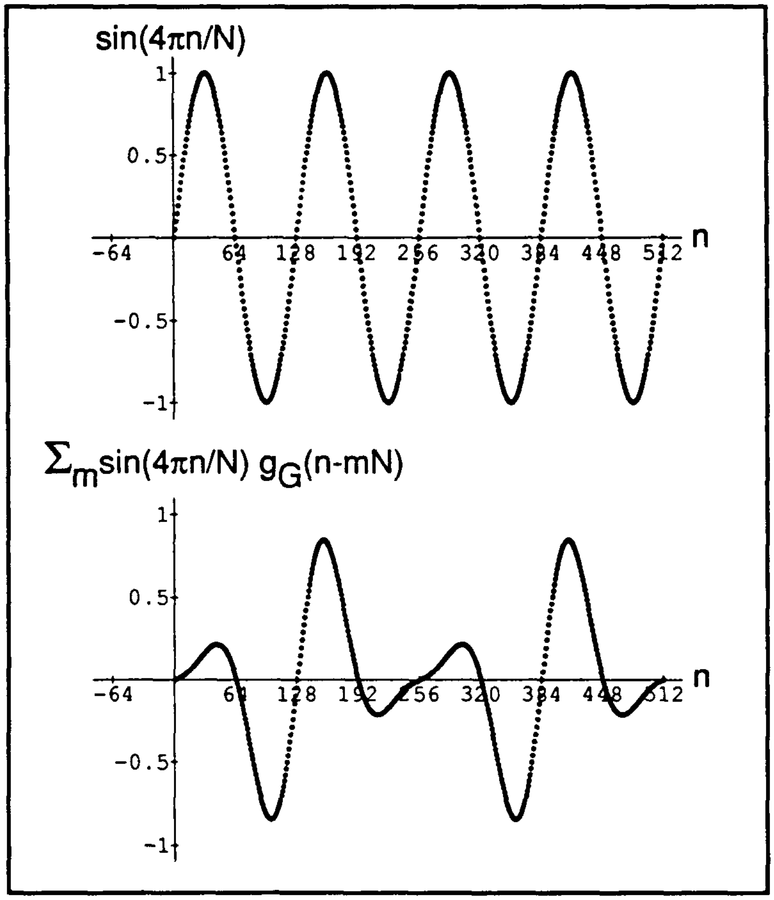

Figure 12. Effect of Windowing with non-Rectangular Functions

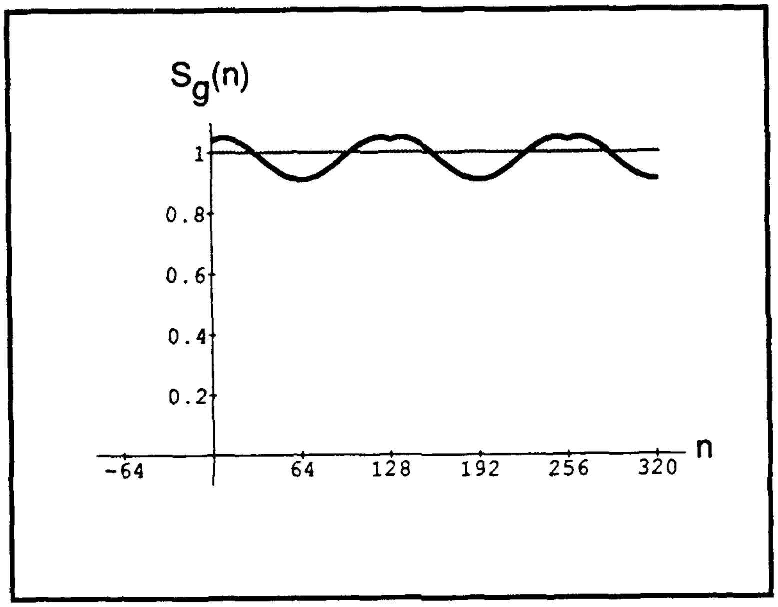

Figure 13. Sum of Gaussian Windows with 50% Overlap

window is used. Figure 15 shows the plot of $S_h(n) = \sum_r g_H(n - rR)$, with $R/N = 0.50$. As can be seen from this figure, the peak error in this case—approximately 5.6×10^{-3}—is less than the peak error produced by the Gaussian window with 25% overlap. It is known that the error produced in the speech signal—by processing the signal in the manner just described—is barely perceptible if a Hamming window with 50% overlap is used. The result of using the compactly supported Gaussian window in this process is investigated next chapter.

The *overlap and add* method described above is more precisely defined by the following transform pair:

$$Y(r,k) = \sum_{n=0}^{N-1} y(n)\, g(n - rR) \exp[-j2\pi kn/N],\ 0 \le k \le N-1 \qquad (34)$$

$$\tilde{y}(n) = 1/N \sum_{r=0}^{N/R(M-1)} \sum_{k=0}^{N-1} Y(r,k) \exp[j2\pi kn/N],\ 0 \le n \le N-1 \qquad (35)$$

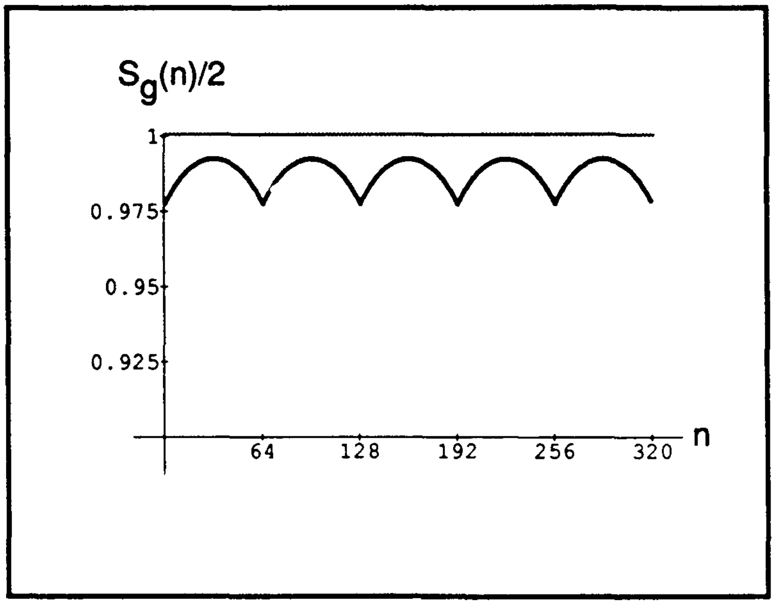

Figure 14. Sum of Gaussian Windows with 25% Overlap

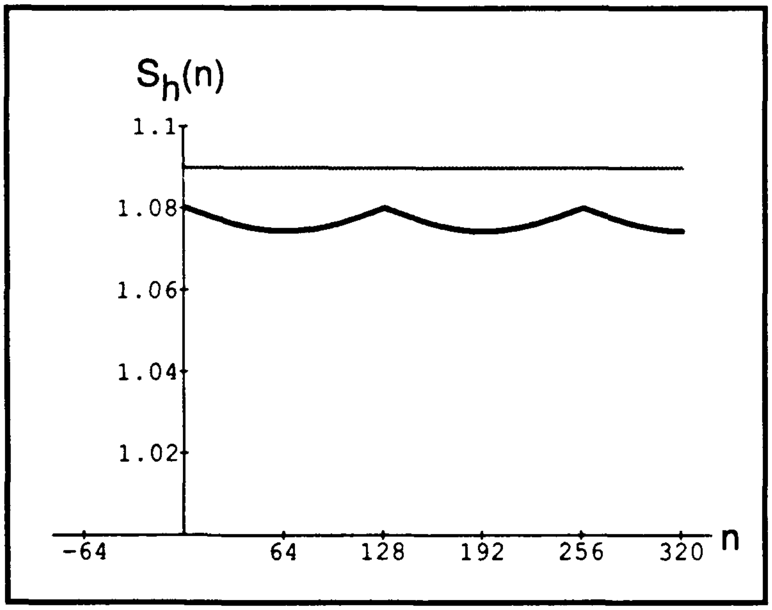

Figure 15. Sum of Hamming Windows with 50% Overlap

where $\tilde{y}(n)$ denotes the approximation of $y(n)$. In this research, $R/N = 0.50$ is chosen in all cases.

One last modification must be made for reconstructing the speech signal from the compressed Fourier components. Before defining this modification, the consequence of compressing in the frequency space is considered. Theory states that in order to avoid *time aliasing*, $Y(m,k)$ must be uniformly sampled at N frequencies for each interval m [58:270]. The reconstruction methods defined above satisfy this condition. However, as previously mentioned, compression in this thesis amounts to eliminating many frequency components in each defined interval of the signal sequence. What remains is an undersampled version of $Y(m,k)$ with nonuniformly spaced frequencies. Therefore, time aliasing is to be expected. As shown in the next chapter, the resulting time aliasing manifests itself as noise or amplitude errors concentrated around the window ends of the synthesized speech. These errors can be attenuated by multiplying the reconstructed intervals of $y(n)$ by a Hamming window. The Hamming window is a good choice since it attenuates the signal at the ends of the window. Also, as was discussed previously, the Hamming window, when overlapped by 50%, produces very little distortion in the signal. The modification leads to the following reconstruction equation:

$$\hat{y}(n) = K \sum_{r=0}^{2M-1} \sum_{k=0}^{N-1} \hat{Y}(r,k)\, g_H(n - rN/2) \exp[j2\pi kn/N],\ 0 \le n \le N-1 \qquad (36)$$

where K is some constant, and $\hat{y}(n)$ is the approximation of $y(n)$ that results from the compressed short-time Fourier coefficients $\hat{Y}(r,k)$.

Since $\hat{Y}(r,k)$ is complex, Eq (36) can be expressed in the following way:

$$\hat{y}(n) = K \sum_{r=0}^{2M-1} \sum_{k=0}^{N-1} \left|\hat{Y}(r,k)\right| \exp[j\phi_{r,k}]\, g_H(n - rN/2) \exp[j2\pi kn/N], \quad 0 \le n \le N-1 \qquad (37)$$

where $\phi_{r,k}$ is the phase angle of $\hat{Y}(r,k)$.

Furthermore, since $y(n)$ is real going into the transform, the regenerated approximation of $y(n)$ should be real coming out. This means that the imaginary terms in Eq (37) can be omitted. This is done by combining the exponential terms, applying Euler's identity, and eliminating the resultant imaginary term, resulting in

$$\hat{y}(n) = K \sum_{r=0}^{2M-1} \sum_{k=0}^{N-1} \left|\hat{Y}(r,k)\right| g_H(n - rN/2) \cos(2\pi kn/N + \phi_{r,k}), \ 0 \leq n \leq N-1 \quad (38)$$

This is the form of the reconstruction formula used by [1, 4, 49] in their investigations of speech coding and noise reduction using a subset of the short-time Fourier space.

To summarize, the equations used to decompose and reconstruct the signal sequences depends on the window used and whether the short-time Fourier coefficients have been compressed or not. If the window function is rectangular and compression is not performed, then Eqs (32) and (33) are used. For Gaussian and Hamming windows, the valid transform pair is described by Eqs (34) and (35) if the Fourier coefficients are not compressed. When compression is performed, Eqs (34) and (36) are used regardless of the window used to compute $Y(r,k)$.

Gabor Decomposition/Reconstruction

For the Gabor decomposition of $y(n)$, Bastiaans' solution (see Chapter II) is adopted here. In general, the discrete version of Eq (6) can be represented as (see [19])

$$C_{m,k} = \sum_{n=0}^{N-1} y_{m,n} \exp[-j2\pi kn/N], \ 0 \leq k \leq N-1 \quad (39)$$

where

$$y_{m,n} = \sum_{q=0}^{Q-1} y(n+qN)\gamma^*(n+qN-mN), \ 0 \leq m \leq M-1, 0 \leq n \leq N-1 \quad (40)$$

where $Q \leq M$. The parameter Q is a free variable in Eq (40) and determines the interval of support of γ. The consequence of varying Q is discussed next chapter. Eq (39) can be conveniently computed using the FFT.

The signal $y(n)$ can be recovered from $C_{m,k}$ by

$$y(n) = \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} C_{m,k}\, g(n - mN) \exp[j2\pi kn/N], \quad N_0 \le n \le 2N - 1 \tag{41}$$

where N_0 depends on the window function $g(n)$.

Eq (41) can be computed using the FFT algorithm. Nevertheless, another method is used in this effort. By similar arguments used in the previous section, Eq (41) can be rewritten as

$$y(n) = \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} |C_{m,k}|\; g(n - mN) \cos[2\pi kn/N + \phi_{m,k}], \quad N_0 \le n \le 2N - 1 \tag{42}$$

Eq (42) is inconvenient to program in the computer since the cosine term must be computed each time for every different phase angle that results. This takes considerable processor time. It is more desirable to precompute sine and cosine tables and access them from memory as needed. Fortunately, Eq (41) can be rewritten in an equivalent form of Eq (42) that when programmed can take advantage of precomputed tables. That form is

$$y(n) = \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} [Re(C_{m,k}) \cos(2\pi kn/N) - Im(C_{m,k}) \sin(2\pi kn/N)]\, g(n - mN), \quad N_0 \le n \le 2N - 1 \tag{43}$$

where

$$Re(x) = \frac{x + x^*}{2} \tag{44}$$

$$Im(x) = \frac{x - x^*}{2j} \tag{45}$$

Two window functions are investigated in the Gabor expansion/regeneration of $y(n)$. These are the sampled versions of the Gaussian window, defined in Eq (10), and the one-sided exponential window, defined in Eq (12). The discrete versions of these signals, in the form used in this research effort, along with their corresponding biorthogonal functions are defined as

Gaussian:

$$g_G(n) = \begin{cases} 2^{1/4}\exp[-\pi(n/N)^2], & -2N \leq n \leq 2N-1 \\ 0, & \text{otherwise} \end{cases} \tag{46}$$

Biorthogonal of $g_G(n)$:

$$\gamma_G(n) = \begin{cases} \varepsilon_1(n)\sum_{i=0}^{2}\varepsilon_2(n,i), & -QN \leq n \leq QN-1 \\ 0, & \text{otherwise} \end{cases} \tag{47}$$

where

$$\begin{aligned} \varepsilon_1(n) &= (1/2)^{1/4}(K_0/\pi)^{-3/2}\exp\left[\pi\,(n/N)^2\right] \\ \varepsilon_2(n,i) &= (-1)^{\lfloor |n/N|+1/2\rfloor+i}\exp\left[-\pi\left(\lfloor |n/N|+1/2\rfloor+i+1/2\right)^2\right] \\ K_0 &= 1.85407468 \end{aligned} \tag{48}$$

Exponential:

$$g_E(n) = \begin{cases} \exp[-n/N]\,u(n), & 0 \leq n \leq 2N-1 \\ 0, & \text{otherwise} \end{cases} \tag{49}$$

Biorthogonal of $g_E(n)$:

$$\gamma_E(n) = \exp[n/N][-u(n+N)+2u(n)-u(n-N)] \tag{50}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x, (e.g., $\lfloor 3.7 \rfloor = 3$). Eqs (46) and (49) define N_0. In the case g_G is used in the transform, $N_0 = -2N$; when g_E is used, $N_0 = 0$. In addition, Eq (50) implies $Q = 1$ for the exponential window's biorthogonal function. The sum in Eq (11), which describes the continuous biorthogonal function of the Gaussian function, converges very rapidly. This is reflected in Eq (47), where the sum is iterated over only three values of the variable i.

As in the case of the short-time DFT, when the Gabor coefficients $C_{m,k}$ are compressed, the overlap and add method is used to suppress the noise that results from time aliasing. Thus, the Gabor coefficients, with R/N=0.50, are found by

$$C_{r,k} = \sum_{n=0}^{N-1} y_{r,n} \exp[-j2\pi kn/N], \; 0 \le k \le N-1 \tag{51}$$

where

$$y_{r,n} = \sum_{q=0}^{Q-1} y(n+qN)\,\gamma^*(n+qN-rN/2), \; 0 \le r \le 2M-1, \; 0 \le n \le N-1 \tag{52}$$

A few steps are needed to obtain the approximation of the signal sequence $\hat{y}(n)$. The approach taken is to reconstruct two partial sequences of $\hat{y}(n)$—each a shifted version of the other by $N/2$. Each sequence is then windowed by repeatedly shifted versions g_H, and finally added together—when properly aligned in time—to produce $\hat{y}(n)$. More explicitly, the partial sequences are defined as

$$\tilde{y}_1(n) = \sum_{r=0}^{\frac{M-1}{2}} \sum_{k=0}^{N-1} \left[Re(\widehat{C}_{2r,k}) \cos(2\pi kn/N) - Im(\widehat{C}_{2r,k}) \sin(2\pi kn/N)\right] g(n-rN),$$
$$N_0 \le n \le 2N-1 \tag{53}$$

and

$$\tilde{y}_2(n) = \sum_{r=0}^{\frac{M-1}{2}} \sum_{k=0}^{N-1} \left[Re(\widehat{C}_{2r+1,k}) \cos(2\pi kn/N) - Im(\widehat{C}_{2r+1,k}) \sin(2\pi kn/N)\right] g(n-rN),$$
$$N_0 \le n \le 2N-1 \tag{54}$$

where $\widehat{C}_{r,k}$ are the compressed Gabor coefficients. Each partial sequence is now windowed by the Hamming function as follows:

$$\hat{y}_1(n) = \sum_{m=0}^{M-1} \tilde{y}_1(n)\, g_H(n-mN) \tag{55}$$
$$\hat{y}_2(n) = \sum_{m=0}^{M-1} \tilde{y}_2(n)\, g_H(n-mN) \tag{56}$$

Finally, the desired approximation of $y(n)$ is obtained by

$$\hat{y}(n) = \hat{y}_1(n) + \hat{y}_2(n - N/2) \tag{57}$$

Affine Wavelet Decomposition/Reconstruction

The discrete wavelet transform can be implemented in various ways. In this effort, the straight forward inner product approach is used. This approach simply computes the wavelet coefficients by taking inner products of the signal with translated and constricted versions of the mother wavelet. The signal is reconstructed by the weighted sum of the wavelets at all shifts and resolutions. The weights are the previously computed coefficients.

A more computationally efficient algorithm was developed by Mallat [44]. In that study, the original wavelet is characterized by two discrete filters. One filter, denoted by G, smooths the signal, thereby filtering frequencies higher than the frequencies of the current level of analysis. The second filter, denoted by H, is associated with the wavelet function and is derived by taking the desired number of derivatives of the smoothing function. This multi-stage process of low-pass filtering followed by higher order derivative filter correlations has been used in the past in spatial image [46] analysis and has also recently been used to develop an orthogonal polynomial transform [47]. Although less efficient, the more direct approach is embraced here for educational purposes and to establish a baseline for any future research.

With that in mind, the discrete versions of Eqs (15) and (16) are used. In general, these equations are defined as

$$\begin{aligned} (Wy)(k, 2^l) &= \sum_{n=-\infty}^{\infty} y(n)\, 2^{l/2} \psi^*(2^l n - k) && (58) \\ y(n) &= \sum_{l=0}^{\infty} \sum_{k=-\infty}^{\infty} (Wy)(k, 2^l)\, 2^{l/2} \psi(2^l n - k) && (59) \end{aligned}$$

where $l, k \in \mathcal{Z}$. The parameter 2^l used in the notation for the wavelet transform, $(Wy)(k, 2^l)$, is adopted here to specify the $\log_2$ analysis performed by the transform.

Since in practice both signal and wavelet sequences are of finite length, Eqs (58) and (59) must be modified. There are other factors to consider that determine the final form of these equations. One of these factors is the mother wavelet to be used. Two mother wavelets are investigated here: the Haar wavelet, defined earlier, and the Morlet wavelet [51]. These are two very different types of wavelet. The sampled Haar wavelet is a real valued function, which is compactly supported in the interval $0 \leq n \leq N-1$. On the other hand, the sampled Morlet wavelet is a complex valued function centered around $n = 0$, which decays to zero at $\pm\infty$. However, for practical reasons, the sampled Morlet wavelet must be truncated. These mother wavelets are defined as

Haar:

$$\psi_H(n) = \begin{cases} 1, & 0 \leq n < N/2 \\ -1, & N/2 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases} \tag{60}$$

Morlet:

$$\psi_M(n) = \begin{cases} \left(\exp\left[-j\omega_0 n/N\right] - C\right) \exp\left[-(n/N)^2/2\right], & -QN \leq n \leq QN-1 \\ 0, & \text{otherwise} \end{cases} \tag{61}$$

where

$$\omega_0 = \pi(2/\ln 2)^{1/2} \tag{62}$$

$$C = \exp\left[-\omega_0^2/2\right] \tag{63}$$

The plots of the real and imaginary portions of the Morlet wavelet are shown in Figure 16. As can be seen from Eq (61) and Figure 16, the mother Morlet wavelet is nearly the Gabor window function modulated at ω_0. One can, therefore, think of the Morlet wavelet as the affine counterpart of the W-H frame Gabor wavelet.

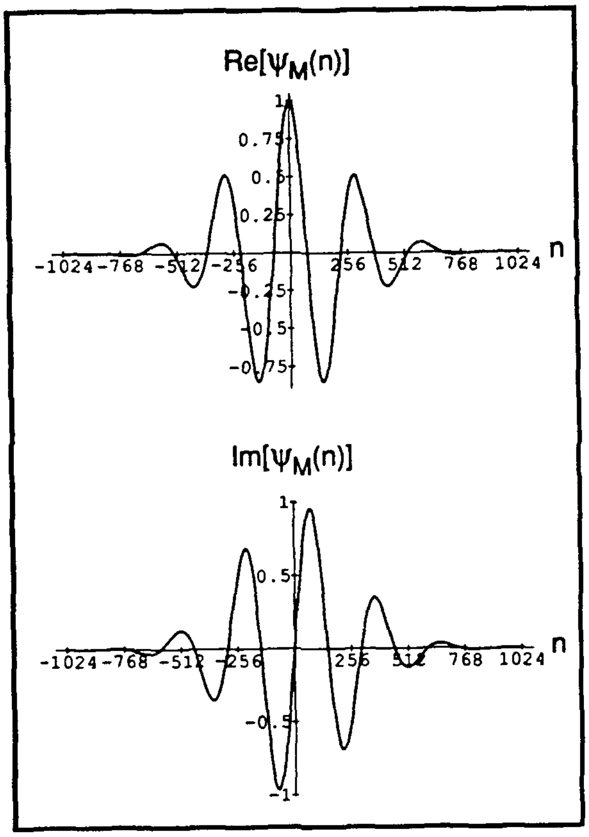

Figure 16. The Morlet Wavelet

The other factors that remain to be determined are the number of samples N, and the total number of levels L necessary to cover the required frequency range of $y(n)$ during the wavelet analysis. When the Haar wavelet is used, $N = 256$ is chosen. This relates to an analysis at level $l = 0$ of approximately 31.3 Hz; the same as the fundamental frequency of the Fourier and Gabor analyses. Only 8 levels ($L = 8$) of analysis are possible; however, these are sufficient to cover the frequency range of $y(n)$. The Haar wavelet analysis using 8 levels produces 255 real valued coefficients for each window of N points. This is one less than the number of useful coefficients found in the short-time Fourier and Gabor decompositions discussed previously.

Morlet and his associates [51:228] suggested that a complete representation of a signal can be computed using four wavelets per octave. The completeness of the set used was not proven, but hypothesized experimentally. This clearly produces many more coefficients than in the radix 2 wavelet analysis proposed here. The tradeoff made here is, obviously, fidelity of the signal versus the number of coefficients used. If the total number of coefficients used in the Morlet case are going to approximate the number of coefficients used in the other analysis/synthesis methods, then the number of analysis levels have to be limited. Consequently, two values of N are investigated . For $N = 256$, the wavelet analysis is computed at octaves of $31.30 f_0$ Hz $\approx$ 26.58 Hz, where $f_0 = \omega_0 / 2\pi$. In this case, 8 levels nearly cover the frequency range of $y(n)$. $N = 128$ is also investigated. The analysis for this value is nearly the same as when $N = 256$; however, the frequency analyzed at $l = 0$ is 53.17 Hz. Only 7 levels of analysis are possible when $N = 128$. Table 1 summarizes the frequencies analyzed–as defined by the parameter N—at the different levels of the wavelet analysis. All values in the table are in Hz. One may wonder why $f_0 = 1$ is not chosen in order to analyze the same frequencies as in the Haar wavelet analysis. For $f_0 = 1$, the imaginary portion of the wavelet—at the highest level of resolution ($l = 7$)–would be sampled at the zero crossings. Therefore, at this level, errors might arise since the imaginary portion of the analysis would always be zero. This problem does not arise when the wavelet is defined as in Eq (61).

Table 1. Frequencies Analyzed (in Hz) Using Haar and Morlet Wavelets

	Haar	Morlet	
l	$N = 256$	$N = 256$	$N = 128$
0	31.30	26.58	53.17
1	62.60	53.17	106.34
2	125.20	106.34	212.67
3	250.40	212.67	425.34
4	500.80	425.34	850.68
5	1001.60	850.68	1701.36
6	2003.20	1701.36	3402.72
7	4006.40	3402.72	—

All the necessary information is now available to define the wavelet transform pair. When the Haar wavelet is used, the signal $y(n)$ is projected onto the wavelet basis by

$$(Wy)(k,2^l) = \sum_{n=0}^{N-1} y(n)\, 2^{l/2} \psi_H^*(2^l n - kN),\ 0 \le k \le (2^l - 1)M \qquad (64)$$

and in turn, the signal $y(n)$ can be reconstructed from the wavelet coefficients by

$$y(n) = \sum_{l=0}^{L-1} \sum_{k=0}^{(2^l-1)M} (Wy)(k,2^l)\, 2^{l/2} \psi_H(2^l n - kN),\ 0 \le n \le N-1 \qquad (65)$$

where M is, again, the number of subsequences of length N that describe $y(n)$. In the case the Morlet wavelet is used, the wavelet transform is

$$(Wy)(k,2^l) = \sum_{n=-QN}^{QN-1} y(n)\, 2^{l/2} \psi_H^*(2^l n - kN),\ 0 \le k \le (2^l - 1)M \qquad (66)$$

and the inverse transform is

$$y(n) = \sum_{l=0}^{L-1} \sum_{k=0}^{(2^l-1)M} (Wy)(k,2^l)\, 2^{l/2} \psi_H(2^l n - kN),\ -QN \le n \le QN-1 \qquad (67)$$

These equations are implemented using a linked binary tree data structure in a recursive algorithm. This approach best fits the structure of the equations that describe the affine transform pair.

These transform pairs described do not change when the wavelet coefficients $(Wy)(k, 2^l)$ are compressed via the LIN. Next section describes how the LIN truncates the W-H and affine frequency spaces.

Compression Using Lateral Inhibition

Before describing the compression process performed by LINs, it is instructive to summarize the methods of compression used in the research that most resembles this effort. Several speech compression methods using short-time Fourier analysis are first presented. Research in the analysis and compression of speech in the affine wavelet space is quite new and the literature sparse. Also, the work that has been done so far appears to be inconclusive. However, one method of compression investigated by Mallat [45] is very closely related to the method used here and, therefore, is reviewed in this section.

Review of Fourier Spectrum Compression Methods McAulay and Quatieri [48, 57] developed a method of reproducing intelligible speech based on only a small subset of the frequency components of the speech signal. In their research, these investigators decomposed speech using the short-time Fourier methods similar to those previously described. Compression was achieved by picking spectral peaks in each time slice—usually referred to as a frame in speech processing—of the short-time Fourier spectrum $(Y(m, k))$. The peaks were chosen by a combination of simple peak-picking and a handful of rules devised to match peaks from frame to frame. McAulay and Quatieri [48] reported that anywhere from 16 to 40 spectral components per frame were sufficient to reproduce *good* quality speech. How they rated good was not defined.

McAulay and Quatieri's research inspired several research efforts at the Air Force Institute of Technology (AFIT) in speech coding, compression, and noise reduction. Bashir [4] and Kabrisky et al. [38] developed a system to improve mutilated speech based on the principles developed by McAulay and Quatieri. However, the spectral peaks were chosen somewhat differently. The glottal frequency was determined and the spectrum was then sampled at that rate. Rules were then developed to search on either side of the originally chosen peaks to determine the final candidates. A rule was also developed to differentiate between voiced and unvoiced speech. That system was reported to have "appreciably" increased the the quality of the noisy speech [38].

In another AFIT study, Alenquer [1] investigated that same peak-picking algorithm for speech compression. In that research, it was thought that since the spectrum of speech rolls off at about 6 dB/octave above 625 Hz or so, that only the first N components should suffice to reproduce the speech signal. Again by subjective criteria, $N = 16$ was found to be sufficient to reproduce good quality speech for male speakers. The system was not tested on female speakers. McMillan [49] has modified the peak-picking strategy further by first designing an equalizing filter that boosts the energy of the higher frequency components to match those of the lower frequency components. The spectrum in each frame is then sampled at the glottal frequency—as in the previous two investigations—however, the N *largest* components are chosen. The rules to search for the final frequency peaks to keep were also expanded. The compressed spectrum is passed through the inverse of the equalizing filter before the speech is reconstructed. No voiced/unvoiced decisions are needed with that system. Currently, seven frequency peaks per frame appear to be sufficient to reproduce good quality speech of either male or female speakers.

The common theme found in these compression algorithms is a search process designed to locate the dominant frequency peaks of the spectrum. All other frequency components are eliminated. The speech, in each of the investigations just reviewed, is reconstructed by summing sinusoids of magnitudes and frequencies corresponding to the chosen frequency peaks.

Review of Affine Wavelet Spectrum Compression Methods Mallat [45] has proven that under certain conditions a complete reconstruction of a signal is possible from only the local extrema of the affine wavelet frequency space. He has also empirically shown that very close approximations to the the original signal can be produced from only the relative maxima of the magnitude of the wavelet coefficients.

The process begins by finding the local extrema of the wavelet coefficients. The eliminated coefficients are then approximated by cubic spline interpolations and the resulting wavelet space sequences are projected onto a convex Sobolev space. The resulting sequences are then projected back out to the signal space, and again into the wavelet space. This procedure is iterated until the original signal is obtained to the desired accuracy. Mallat described a similar process in the case the relative maxima of the wavelet coefficients are chosen.

LIN Design The methods used to compress the Fourier spectral characteristics of speech just reviewed have relied on rule-based systems that have been developed in an ad hoc manner. Specific knowledge regarding the Fourier spectral characteristics of speech was necessary in order to design the systems based on Fourier expansion. As is shown in this section, LINs will replace the rule based systems for compression in both the W-H frequency space and the affine frequency space. In contrast, the design principles of the LINs are based on a solid mathematical foundation and on several known physiological phenomena of hearing.

The design principles used in this thesis are meant for wider application than just compressing the speech spectrum. The choice of LINs for this research is biologically motivated, and as such, specific knowledge about how the hearing mechanism works is used. This approach differs from the approach of the previous efforts, since the emphasis is not placed on knowledge about source signals but on how biological systems process these signals. Animals across species use the basic principles adopted here in nearly all sensory systems to preprocess environmental information. These design principles may, therefore, be applicable to coding and compressing other acoustical and electromagnetic signals as well as two-dimensional spatial images.

In the affine wavelet case, Mallat's compression and reconstruction methods are based on a more solid mathematical foundation than the other compression methods reviewed here. Nevertheless, Mallat's iterative process does not appear to be feasible for real-time processing. As shown in a following section, the local maxima of the wavelet spectrum will be chosen by the LIN but no attempts of regenerating the eliminated coefficients will be made. One of the main reasons for this is to compare the effects of eliminating coefficients in the W-H transform space versus the affine wavelet transform space of speech. The other reason for not using Mallat's iterative algorithm is for computational reasons.

LIN Compression of Short-Time Fourier and Gabor Spectra To recall from Chapter II, LINs are found in the subcortical relay stations that carry patterns of electrical activity from the auditory nerve to the primary auditory sensory fields of the cortex. The main function of the LINs in that system is presumed to be contrast enhancement of spatio/temporal information. This is precisely the function assigned to LINs in this effort. The magnitude of the spectral coefficients of speech are analogous to the electrical activity found in the auditory nerve. The dynamics of the LINs will search for *local* energy peaks and suppress weaker ones. Nothing is assumed about the incoming signal. The only information used to design the competitive architecture of the LINs is the experimentally derived filter bandwidths of the cochlea.

The LIN design for W-H frequency compression begins with the numerical or discrete time approximation of the activity equation of LIN nodes described in Eq (18) in Chapter II. Figure 8, with the indexes relabeled, is shown again in Figure 17 for more convenient reference. The discrete time approximation of Eq (18) used is found by Euler's method, one of the simplest techniques known for solving differential equations. As long as the inputs to the LIN are normalized or scaled between zero and one, Euler's approximation of Eq (18) remains stable throughout its computation, leading to steady-state, with reasonable step size values. Thus, for a given step size τ, the activity equation

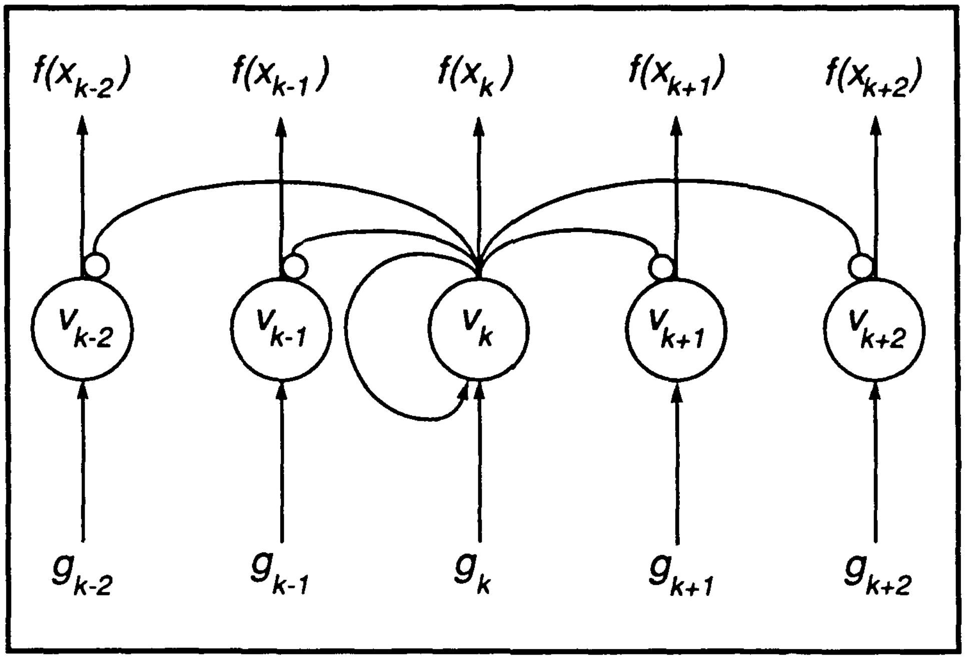

Figure 17. Lateral Inhibition Network Model

becomes

$$x_k(t+1) = x_k(t) + \tau\{[A - x_k(t)]E_k - [B + x_k(t)]I_k\} \tag{68}$$

where

$$\begin{aligned} E_k &= g_k + f(x_k) \\ I_k &= D\sum_{j\neq k} f(x_j) \\ g_k &= |Y(r,k)| \text{ or } |C_{r,k}| \end{aligned}$$

The output, Eq (19), is modified to include a threshold and becomes

$$f(x_k) = \begin{cases} (x_k - \theta)^2/[(x_k-\theta)^2 + \alpha^2], & x_k \geq \theta \\ 0, & x_k < \theta \end{cases} \tag{69}$$

The variables in Eqs (68) and (69) are chosen as $A = 1$, $B = 1$, $D = 1$, $\alpha = 0.5$, and $\tau = 0.1$. In addition, the result of varying θ is investigated in Chapter IV as well.

In simple terms, Eq (68) says to increase the activity x_k of node v_k by a maximum of $A\tau$ if the excitatory term is greater than the inhibitory term ($E_k > I_k$), or decrease x_k by a maximum of $B\tau$ if $I_k > E_k$. In other words, if the current node's input plus feedback from itself is greater than the sum of the outputs of the competing nodes, then that node's activity will increase and will tend suppress the activity of the competing nodes. Parameter D is chosen to produce *winner-take-all* competition. This means that the node with the highest input will kill off the activity of all nodes competing with it. In this way the spectral coefficient with the largest magnitude will be chosen over others within a defined bandwidth.

One can also view Eq (68) as an implementation of a *Mexican hat* function discrete filter. Indeed, the plot of a typical implementation of the transfer function (H_M) of this filter resembles a Mexican hat, as can be seen from Fig (18). This particular filter implementation is modeled by the negative of the second derivative of the Gaussian function, or

$$H_M(k) = [1 - (k/N)^2] \exp[(k/N)^2/2] \tag{70}$$

It is evident from Fig (18) where the term on-center off-surround processing—the alternative expression for lateral inhibition—originated. Frequency components in the center of the filter are magnified, whereas the frequency components on the periphery of the filter are attenuated. In the specific case of the LIN used in this thesis, the filter center boosts the energy of only one spectral component, and uniformly attenuates an equal number of components on either side of the center. These filter bandwidths are now defined.

A *competitive band* is defined to be a set of competing nodes. Similarly, a *competitive bandwidth*, denoted as $W_C \in \mathcal{N}$, is the number of nodes in a competitive band. The competitive bandwidth, therefore, has a frequency bandwidth W associated

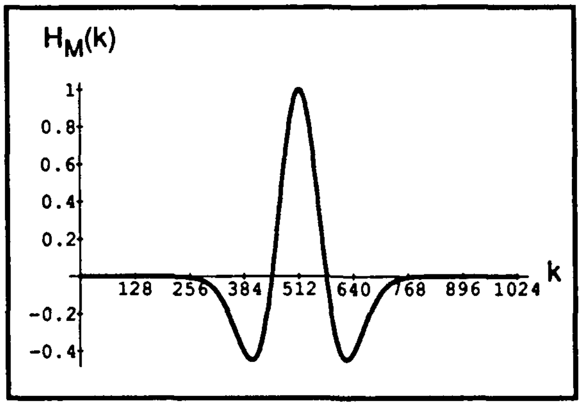

Figure 18. Typical Mexican Hat Filter Function

with it, and is defined by the following relationship:

$$W = 31.3\,W_C \text{ Hz} \tag{71}$$

The LIN is designed to have overlapping competitive bands—or overlapping Mexican hat filters. The competitive bandwidths of these overlapping bands are chosen to approximate the bandwidths of the triangular filter models of the cochlea. As one may recall from Chapter II, the bandwidths of these overlapping filters increase logarithmically as a function of frequency. There are two main models describing these bandwidths. One is called the equivalent rectangular bandwidth (ERB) [50]; the other is called the critical bandwidth (CB) [70].

The ERB was derived to have the following function:

$$W_{ERB}(f) = 6.23f^2 + 93.39f + 28.52 \tag{72}$$

The function ascribed to the CB is

$$W_{CB}(f) = 25 + 75(1 + 1.4\,f^2)^{0.69} \tag{73}$$

where *f* is frequency in kHz. A plot of these two relationships is shown in Figure 19. This plot clearly shows that the ERBs are much more conservative that the CBs, especially for frequencies less than 600 Hz. A LIN designed to approximate the CB model will, therefore, produce more compression.

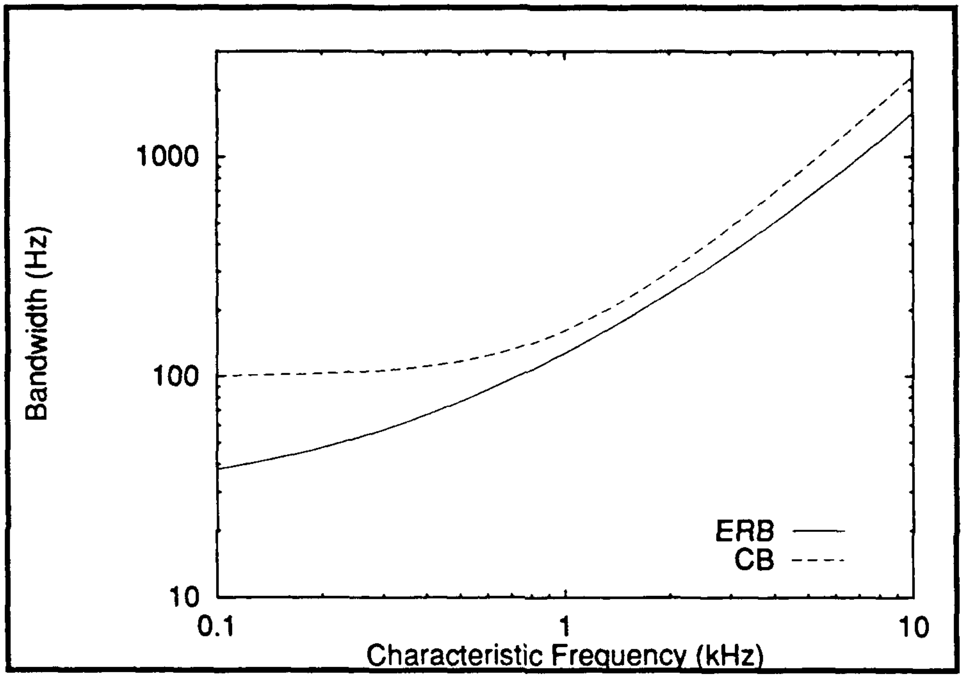

Figure 19. ERB and CB as Functions of Frequency

In practice, the frequency bandwidths associated with the competitive bandwidths of the LIN can only approximate the experimentally derived bandwidths of the cochlear filters. The reason for this is that each LIN node is chosen to compete with an equal number of nodes on each side of it for simplicity. This means that the frequency bandwidths of the LIN cannot increase smoothly, but in steps. Table 2 summarizes the competitive bandwidths W_{C_k}, and the associated frequency bandwidths W_k of the LIN that approximate the ERBs and CBs of the cochlea. The column labeled **ERB(CB) CF** shows the characteristic frequencies, CFs, that have associated ERBs(CBs) equal to that of the frequency bandwidths W_k of the LIN. In other words, $W_{ERB(CB)}(CF) = W_k$.

The column labeled **ERB(CB)** k, shows the frequency indexes k that produce the analog frequencies ν_k closest to the ERB(CB) CFs. Figures 20 and 21 show the LIN's frequency bandwidth approximations of the ERB and CB curves, respectively, as defined by Table 2.

Table 2. Bandwidths (in Hz) Analyzed by LIN

W_{C_k}	W_k	**ERB**			**CB**		
		CF	k	ν_k	CF	k	ν_k
3	93.9	670	21	657.3	—	—	—
5	156.5	1264	40	1252.0	947	30	939.0
7	219.1	1820	58	1815.4	1456	47	1471.1
9	281.7	2344	75	2347.5	1880	60	1878.0
11	344.3	2842	91	2848.3	2262	72	2253.6
13	406.9	3318	106	3317.8	2616	84	2629.2
15	469.5	3773	121	3787.3	2950	94	2942.2
17	532.1	—	—	—	3269	104	3255.2
19	594.7	—	—	—	3575	114	3568.2
21	657.3	—	—	—	3870	124	3881.2

To summarize, the LIN designed to compress the W-H wavelet spectrum can be thought as a series of overlapping Mexican hat filters. The bandwidths of these filters are chosen to approximate the putative filter bandwidths of the cochlea, which increase in bandwidth logarithmically as a function of the CF. The function of these filters is to search for energy peaks in their bandwidth and attenuate all others.

Once the LIN converges, all non-zero output nodes are used as pointers to the original spectrum. This means that the original spectral magnitudes (of the chosen coefficients) are used to resynthesize the signal, and not the values of the outputs of the LIN nodes.

LIN Compression of Affine Wavelet Spectrum The LIN design for compressing the affine wavelet spectrum is based on more speculative criteria than what was used in the previous section. The LIN design for the W-H wavelet spectrum is based on an

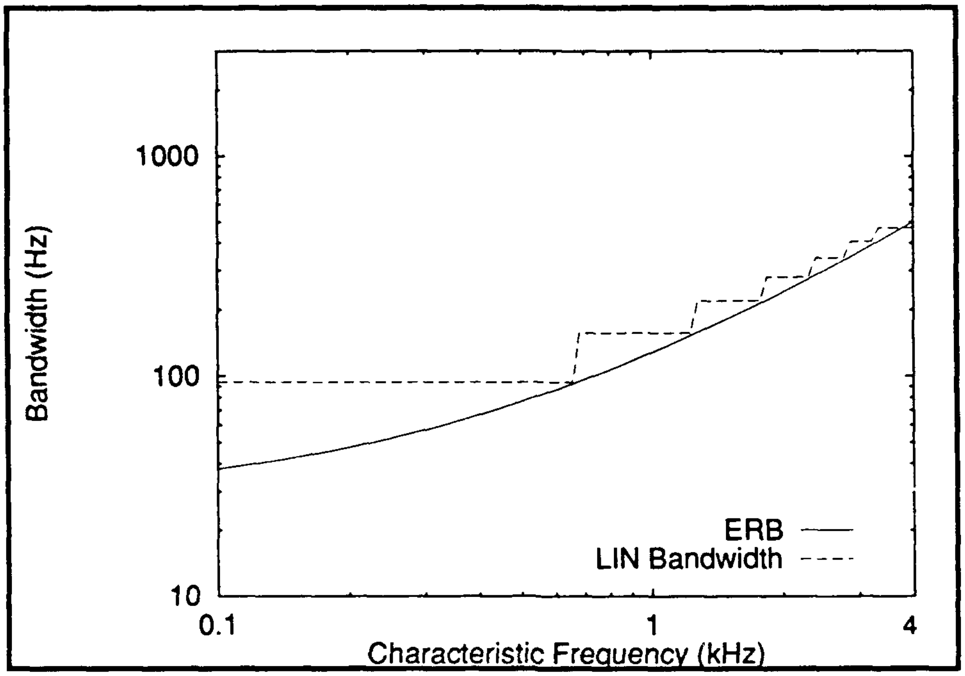

Figure 20. LIN Frequency Bandwidth and ERB vs Frequency

attempt to approximate filter bandwidths found in the cochlea. These filter bandwidths are nearly constant for CFs less than 1 kHz or so, and then increase logarithmically for CFs above 1 kHz. The resultant W-H transform/LIN process is a forced logarithmic frequency analysis of the input signal. This analysis highlights the frequency peaks or changes in each time slice, or frame. The match between each frame of the W-H spectrum and the LIN is straightforward, since both are one-dimensional representations of frequency or frequency operators. In contrast, the affine wavelet transform performs a logarithmic analysis of the input signal to begin with. Each affine wavelet frame—defined in this case as the interval of time of the lowest level analysis—is both a time and frequency representation.

The direct counterpart of the previous design, is to highlight the temporal changes or peaks in each octave band or analysis level. The LIN design for compressing the affine spectrum at each analysis level is identical to the LIN design for compressing the

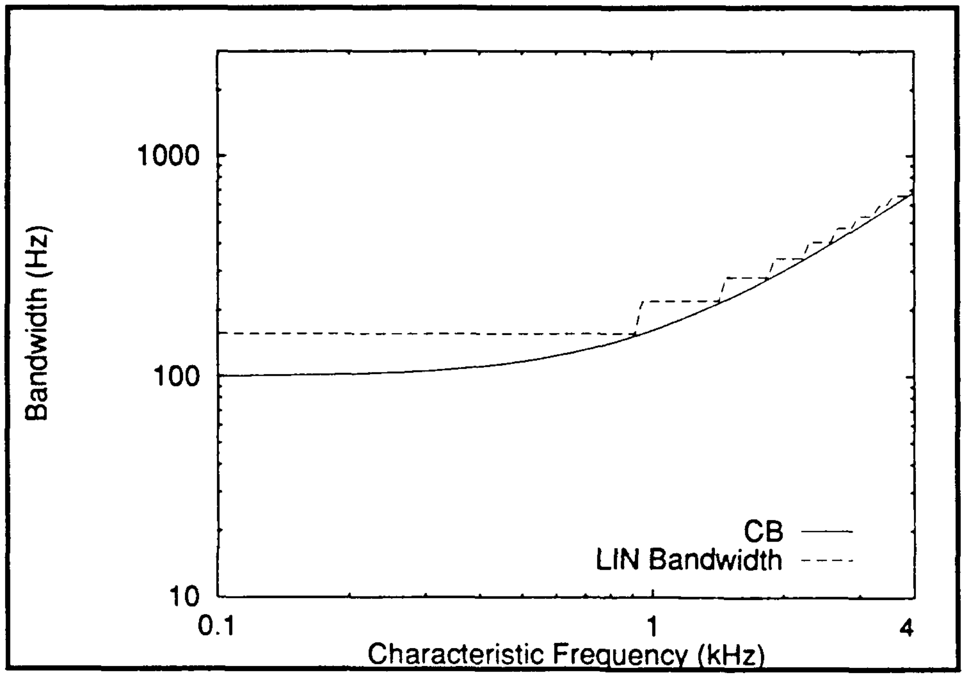

Figure 21. LIN Frequency Bandwidth and CB vs Frequency

W-H spectrum; however, the competitive bands remain constant for any given level. A competitive band in the present case now corresponds to a competitive interval, which is denoted as $T_C \in \mathcal{N}$. The relationship between the competitive interval and the true interval associated with it in this effort is

$$T_l = \frac{31.95\, T_C}{2^l}\ \text{msec.} \tag{74}$$

where l is the level of analysis, and the time constant 31.95 msec. was found in the beginning of this chapter.

The competitive interval for each level is determined by the reciprocal of competitive bandwidth for the frequencies analyzed by each level of analysis. In this case, only the CB rule is used. Table 3 lists the intervals over which the LIN is compressing at each level. These approximate the intervals determined by the CB rule, which are denoted by T_{CB} and are also listed in Table 3.

Table 3. Intervals (in msec.) Analyzed by LIN

l	T_C	T_l	T_{CB}
0	0	0	9.99
1	0	0	9.97
2	0	0	9.89
3	3	11.98	9.57
4	5	9.98	8.52
5	5	4.99	6.15
6	7	3.49	3.32
7	7	1.75	1.45

Table 3 shows that no compression is possible in the first three levels of analysis using the CB tuned LIN method of compression. This suggests that the affine decomposed speech will not be compressed to the levels that the W-H decomposed speech is capable of being compressed with LINs.

Summary

This chapter defined the decomposition, compression, and reconstruction algorithms used in this thesis. The chapter began with a description of the software and hardware environments along with some basic definitions and notation. The short-time Fourier, Gabor, and affine wavelet transforms, along with the varied window or wavelet functions used, were then precisely defined. Finally, the LIN designs for compressing each of the transform derived spectra were presented. Briefly, the LINs' overlapping bandwidths(intervals) were designed to approximate the bandwidths(intervals) of the theoretical overlapping filter model of the cochlea. The following chapter summarizes the results found in regenerating speech signals from the compressed spectra obtained from the methods described in this chapter.

IV. Results

Introduction

This chapter summarizes the results obtained from decomposing, compressing, and regenerating the spoken integers by the methods described in the previous chapter. For economy, the results are shown only for the word *seven* spoken by the author unless otherwise noted. These results are representative of the results obtained from the entire vocabulary.

This chapter is organized into four main sections. In the first section, the results obtained from compressing the short-time Fourier and Gabor spectra are discussed. These results highlight the differences obtained from using the various window functions and various LIN configurations. The second section summarizes the results obtained from compressing the affine wavelet spectrum. The third section discusses the results obtained from the noise filtering tests. The final section compares the results summarized in previous sections.

The criteria used here to establish the *best* results are mean square error (MSE) between the original time signal and the reconstructed signal (both signals are amplitude normalized before the MSE is computed), and subjective listening tests. The subjective tests consist simply of questions regarding the intelligibility and quality of the reconstructed speech judged by randomly chosen listeners. Since this research is only a preliminary study in the LIN compressing capabilities of a variety of spectra, such informal testing seems reasonable. Any subsequent research based on this thesis that looks to optimize the intelligibility and communication bit rates of speech should use more rigorous listening tests such as the diagnostic rhyme test.

Results Based on Short-Time Fourier and Gabor Spectra

The results reported in this section are organized by first comparing the original discrete signal to the reconstructed signal from all the spectral coefficients produced from the short-time Fourier and Gabor decompositions defined in the previous chapter. The results of compressing the varied spectral representations with the LIN are then examined. These results show how the spectral components are eliminated as the competitive bandwidths are increased and the LIN output threshold is increased. Finally, the reconstructed signals from their compressed spectral representations are compared and ranked by MSE and auditory quality.

The plots of the original and the reconstructed signal sequences of the word *seven*, from the various short-time Fourier and Gabor representations, are shown in Figures 22 through 32. Immediately after each of these plots, the difference signal between the original and reconstructed signals are shown, rescaled, in order to better demonstrate the resultant error. All signals are linearly scaled between minus one and one in order to eliminate irrelevant scale variations.

The plot titles describe the window and the method of decomposition used. *Spectrogram* in these titles refer to the short-time Fourier decomposition, and *Gabor* is self explanatory. For example, the plot labeled Seven Rectangular Spectrogram describes the resynthesized word *seven* derived from the short-time Fourier spectrum using a rectangular window. The plot entitled Seven Original, naturally, is the plot of the original signal. The word *seven* is omitted from the title of the plots illustrated in Figures 31 and beyond, with the understanding that the results shown are derived from this spoken word. All signal sequences in this section are discretized, of course, but are plotted with points joined and give the illusion of continuous time signals.

As may be recalled from the previous chapter, the interval of support of the biorthogonal function is determined by parameter Q in Eqs (40) and (52). That interval of support is defined by the inequality $-QN \leq n < QN$, where N is the number of points in each window. By definition, $Q = 1$ for the exponential biorthogonal function. In the case of the Gaussian biorthogonal function , $Q \leq M$ [19]. Two values of Q were chosen in the Gaussian case: $Q = M$, for the maximum support, and $Q = 2$, for minimum support. The plots labeled Gauss Gabor are of reconstructed signals from the Gabor representation computed using $Q = M$. Likewise, the plots labeled Short Gaussian Gabor are associated with $Q = 2$. This nomenclature will be adopted hereafter for all references to the Gabor representations computed using the two values of Q.

The plots in Figs 22 through 25 reveal that the errors produced from reconstructing the signal from the short-time Fourier spectrum using rectangular or Hamming windows are very small and nearly identical. Taking floating point and other arithmetic errors into account, these small errors are to be expected from the theory described in the previous chapter. The result of using the compactly supported Gaussian window produces a larger error in the reconstructed signal, and again this is to be expected from the theory. Numerous subjects listened to the original and reconstructed signals and could not distinguish between them.

As seen from Figures 28 through 33, the errors produced from the Gabor representations are much greater and different in nature. Figure 28 illustrates the reconstructed signal from the Gaussian Gabor spectrum, that is, using the maximally supported biorthogonal function of the Gaussian window. The first and last few time frames were omitted in the reconstruction process in order to avoid out of range indexing in the data structure used to hold the values of the reconstructed signal. Omitting these time frames created two unexpected spikes at the beginning and end of the reconstructed signal. These errors foreshadow the errors that occur when the Gabor spectrum is compressed. Figure 29 illustrates the error signal produced from the signals plotted in Figure 28. The two large spikes in the reconstructed signal were removed prior to computing that difference. Besides the 31.3 Hz sinusoidal error observed in Figure 28, a handful of impulses are also seen. Although these errors are much greater than the errors produced by the short-time Fourier reconstructions, they were not perceived by the listening subjects.

As shown in Figure 31, the reconstructed signal from the short Gaussian Gabor spectrum has large periodic spikes. These spikes were clearly audible in the listening tests. Although intermediate values of Q were not tested, it is clear that subjective quality of the reconstructed signal is a function of the parameter Q. This is also verified by the MSE of these signals shown in Table 4.

It appears that the spikes that are manifested in the reconstruction process *leak* through when the Gaussian and its biorthogonal function do not entirely cancel each other out. The large spikes, shown in the lower plot in Figure 28, that occurred when sections of the Gabor coefficients were removed support this conjecture.

Figures 32 and 33 illustrate the result of reconstructing the speech signal from the Gabor representation using an exponential window. Figure 33 shows that periodic (31.3 Hz) spiky errors are also produced in this reconstructed signal. The arrows point to the most obvious ones. Also notice that the overall error tends to follow a decaying exponential function; the same as the window function. The explanation for these errors is different from the one offered in the Gaussian case. This explanation is given without proof with the following simple example. Suppose a modulated one sided exponential function is defined with the same time constant as the exponential window used in a Gabor expansion. Furthermore, suppose that this function is exactly shifted in time by an integer multiple of $\alpha = \Delta t$ and modulated by an integer multiple of $\beta = \Delta f$. A Gabor decomposition of this signal will result in a single impulse in the two-dimensional Gabor lattice. This impulse results from the perfect match between the signal and the window shifted and modulated at precisely the same values as the signal. Figure 34 precisely illustrates this phenomenon. The original signal plotted on the top left hand graph is defined as

$$y(t) = \exp[-(t - t_0)/\tau]\,\sin(2\pi f_0 t)\,u(t - t_0) \tag{75}$$

where

$$\begin{aligned} t_0 &= 3 \\ f_0 &= 11 \end{aligned}$$

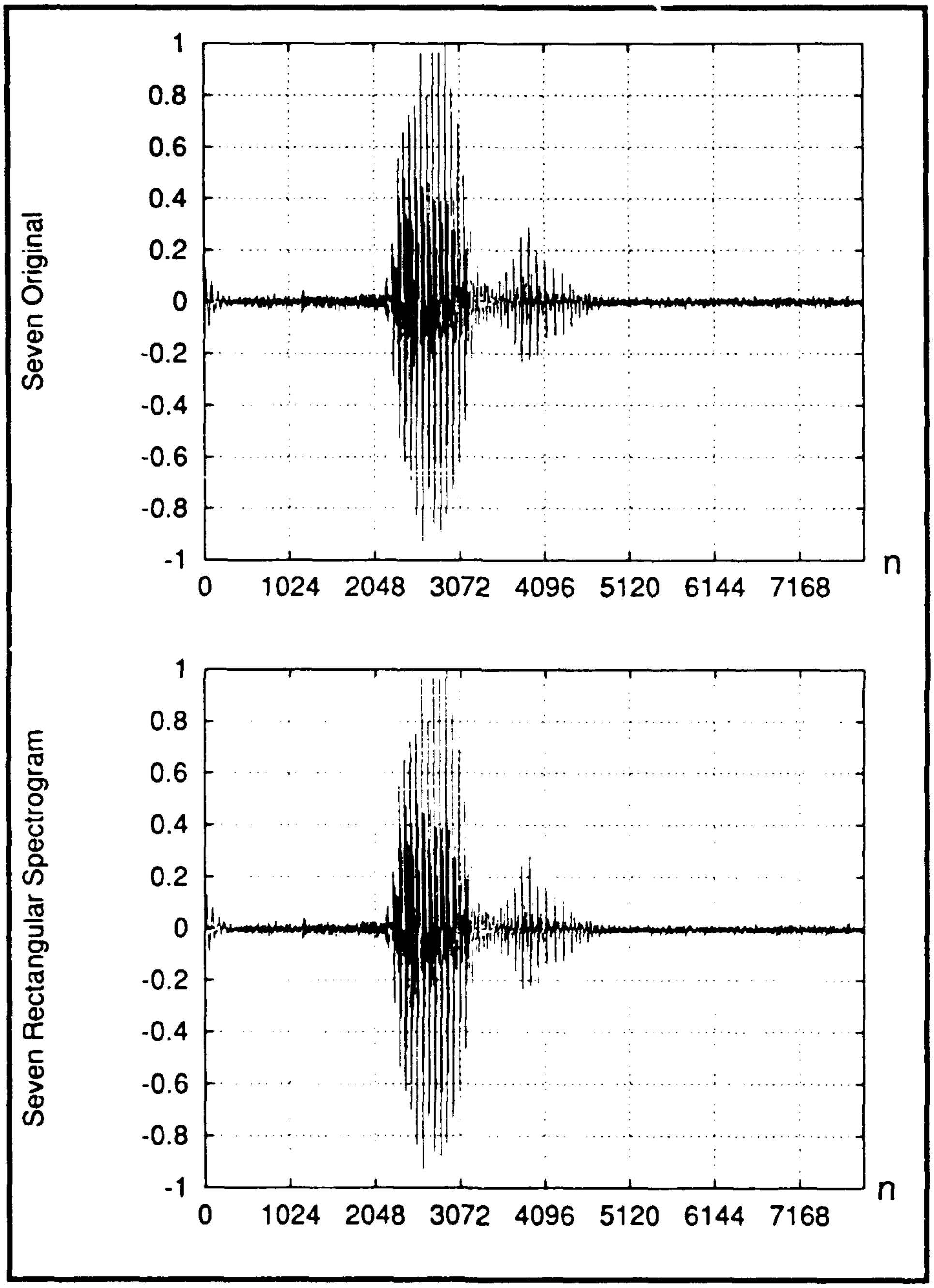

Figure 22. Original Signal and Reconstructed Signal from Spectrogram Using a Rectangular Window

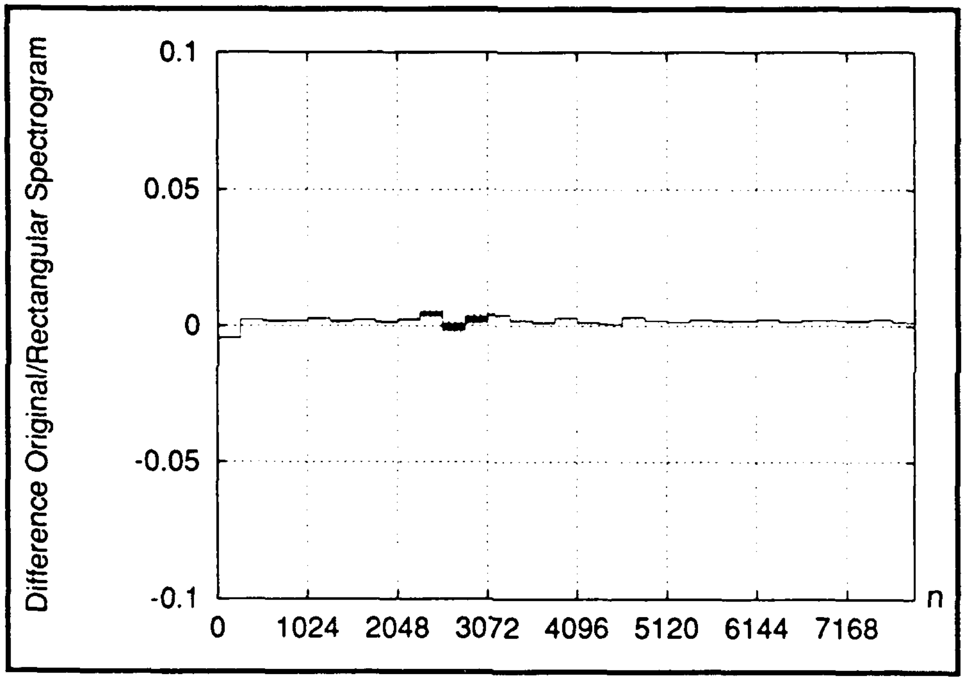

Figure 23. Difference Signal Between Original and Reconstructed from Spectrogram Using a Rectangular Window

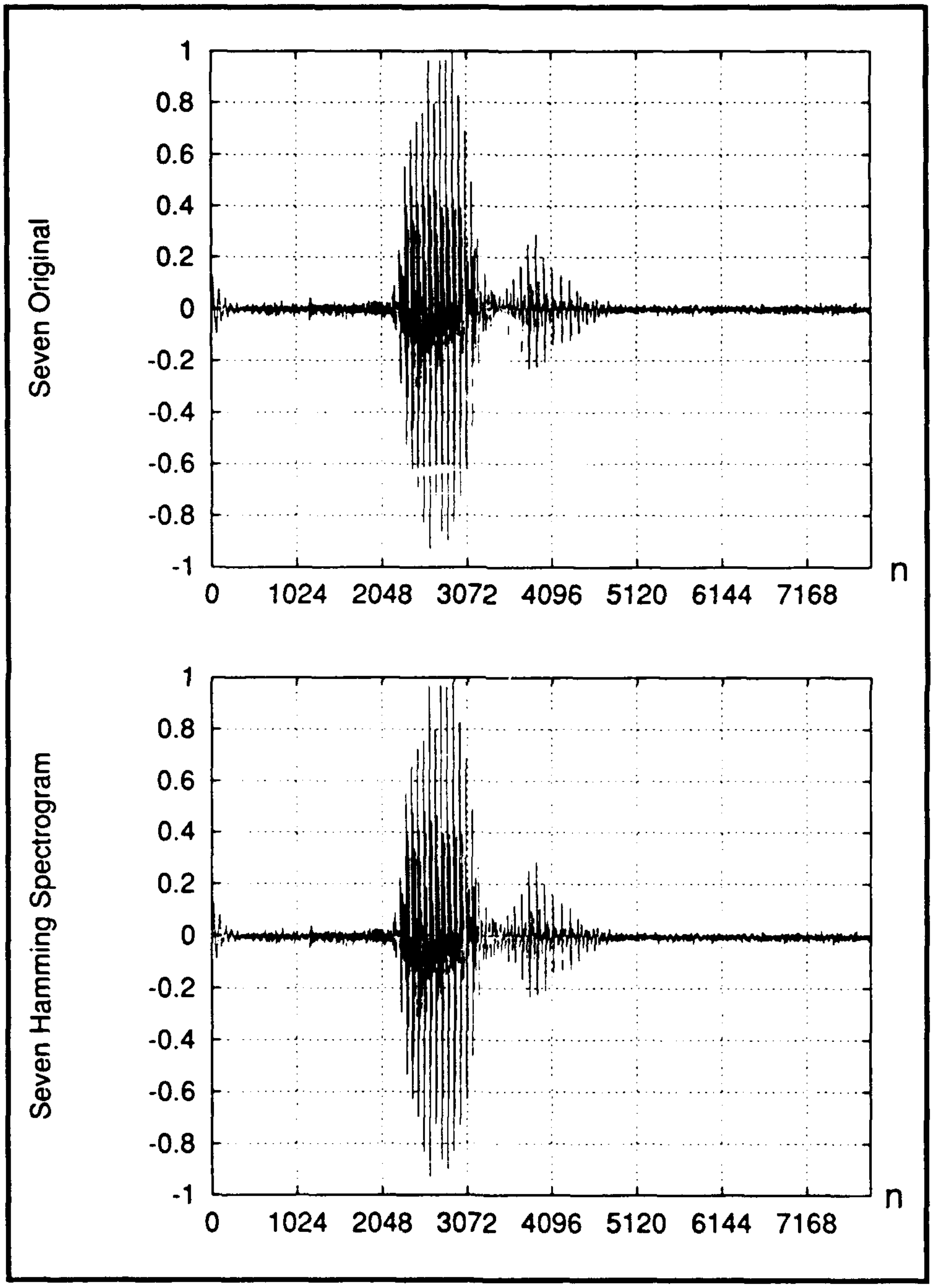

Figure 24. Original Signal and Reconstructed Signal from Spectrogram Using a Hamming Window

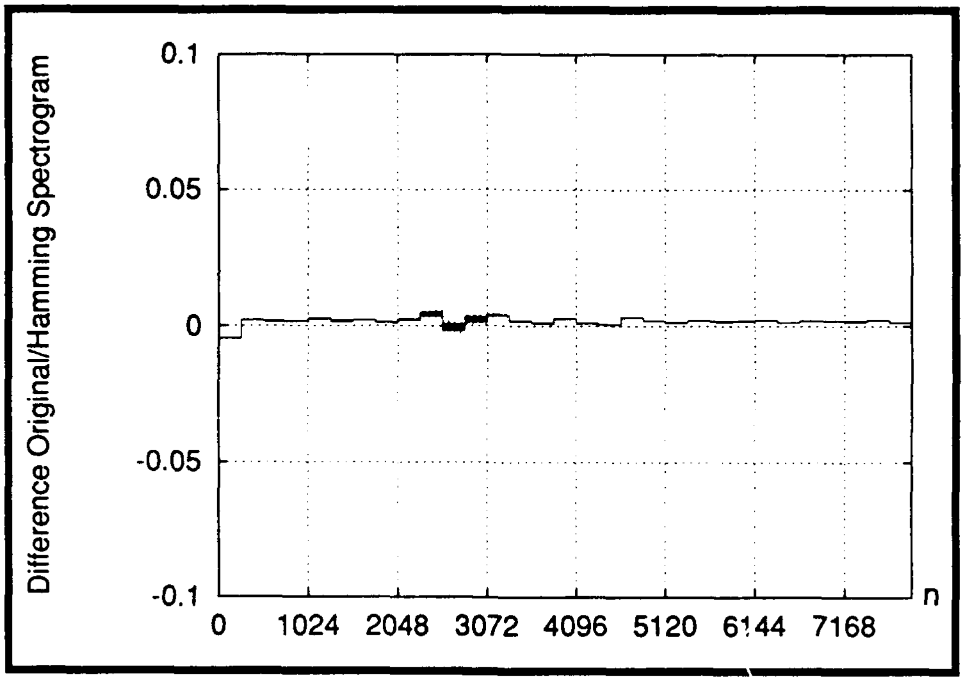

Figure 25. Difference Signal Between Original and Reconstructed from Spectrogram Using a Hamming Window

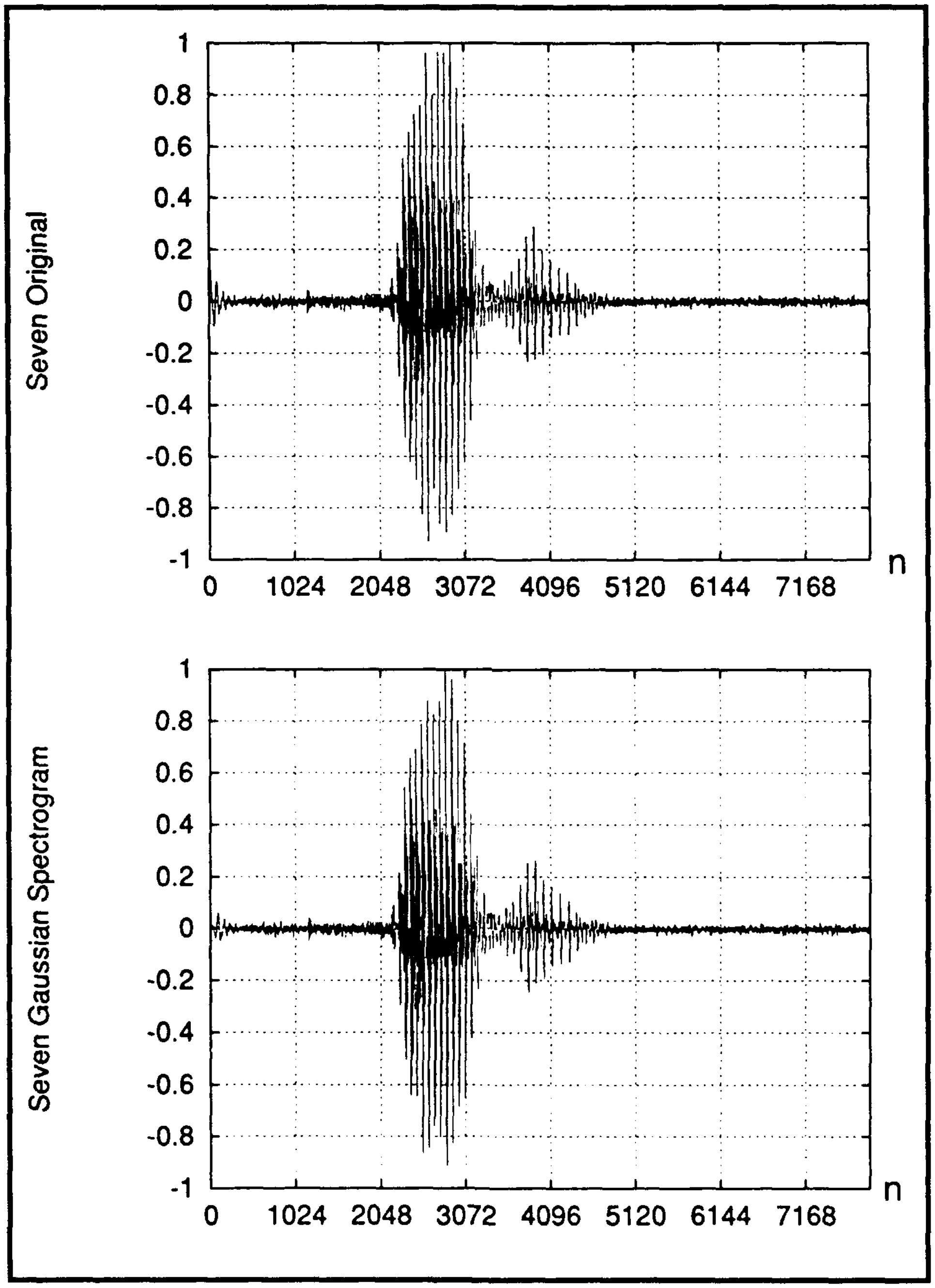

Figure 26. Original Signal and Reconstructed Signal from Spectrogram Using a Gaussian Window

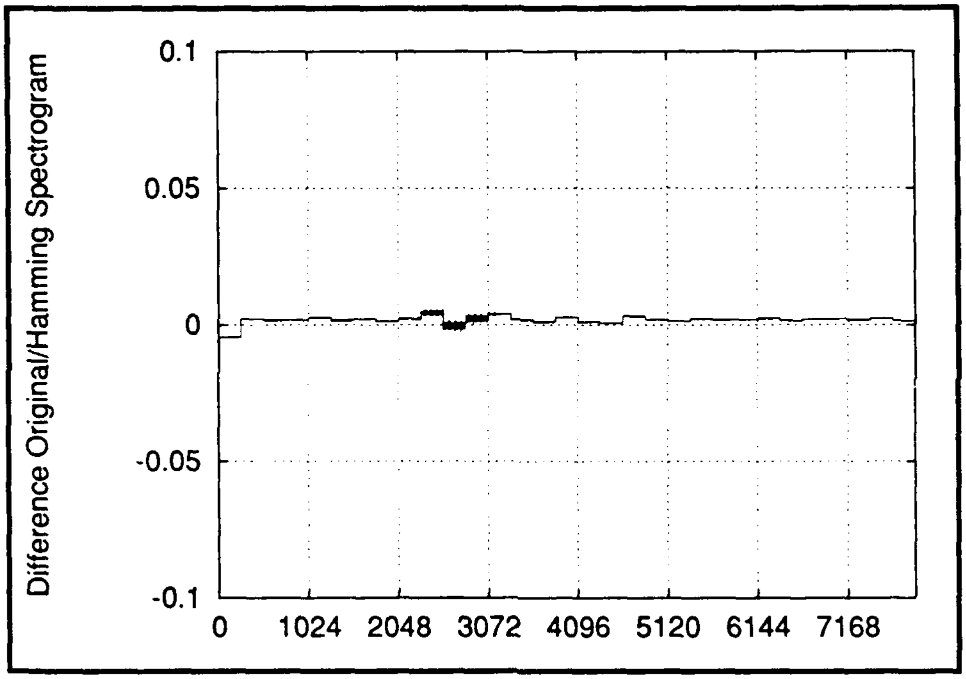

Figure 27. Difference Signal Between Original and Reconstructed from Spectrogram Using a Gaussian Window

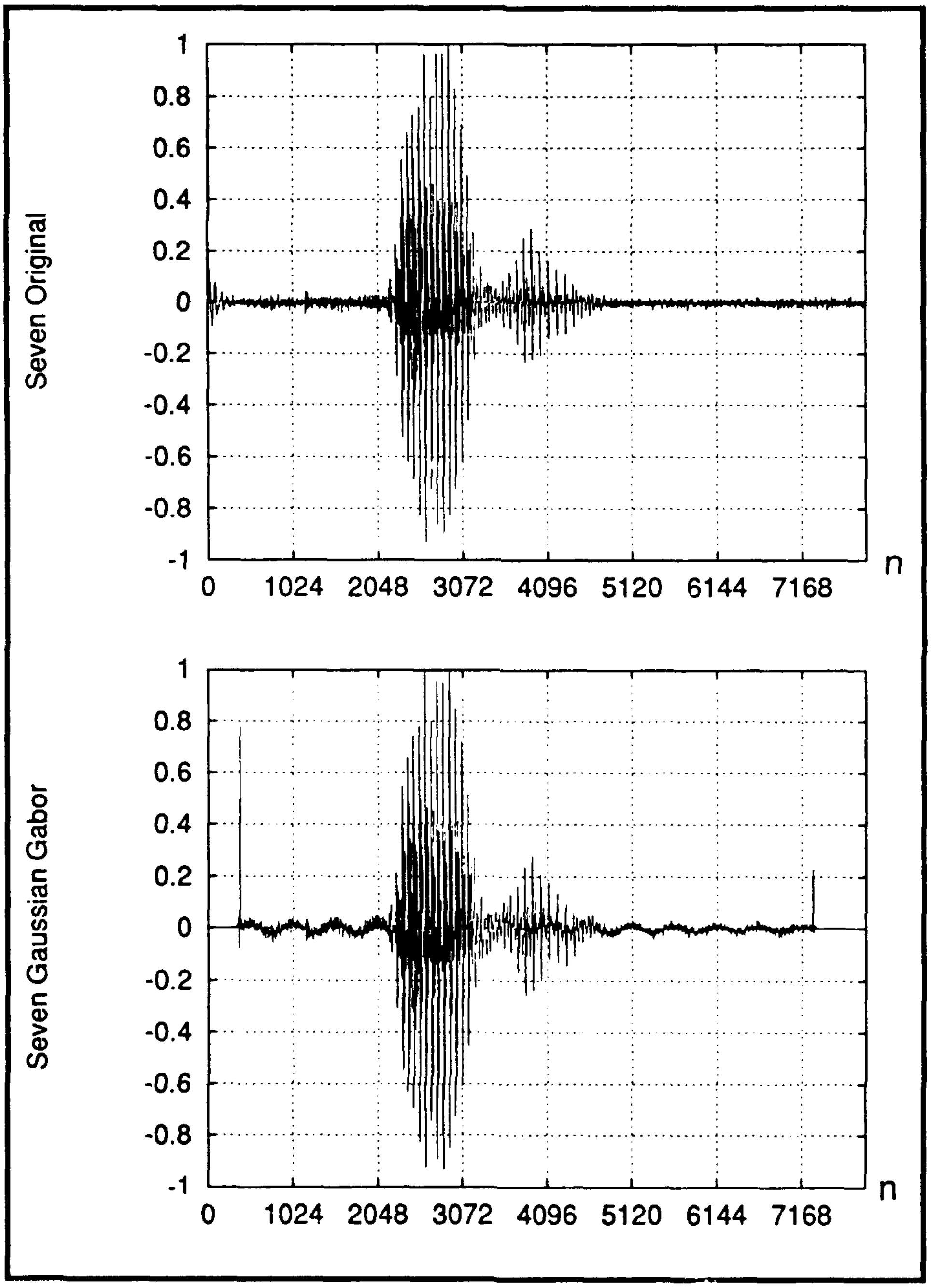

Figure 28. Original Signal and Reconstructed Signal from Gaussian Gabor Spectrum

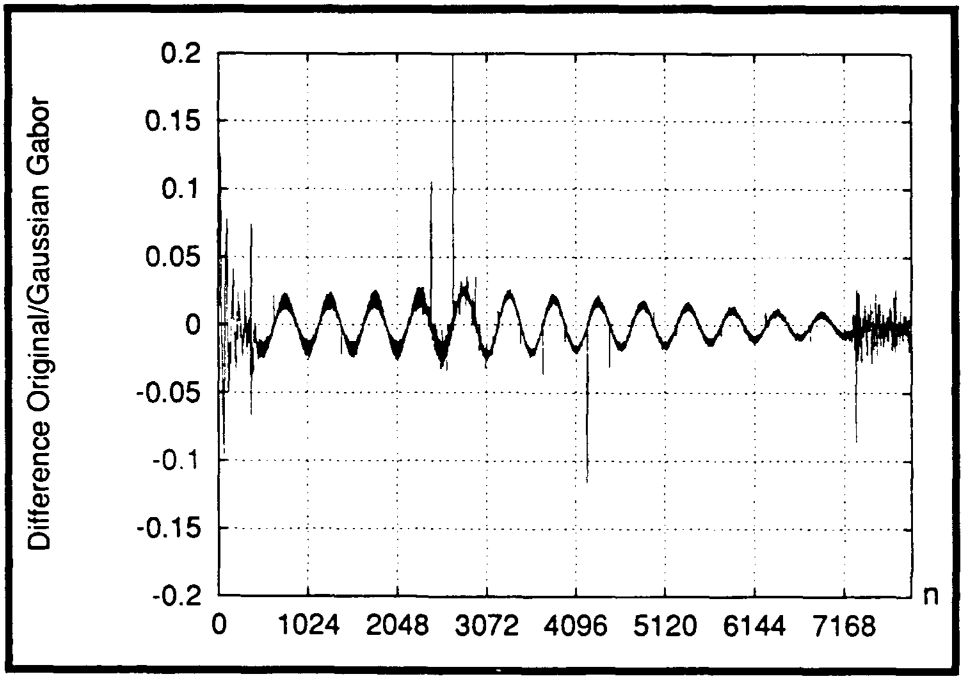

Figure 29. Difference Signal Between Original and Reconstructed from Gaussian Gabor Spectrum

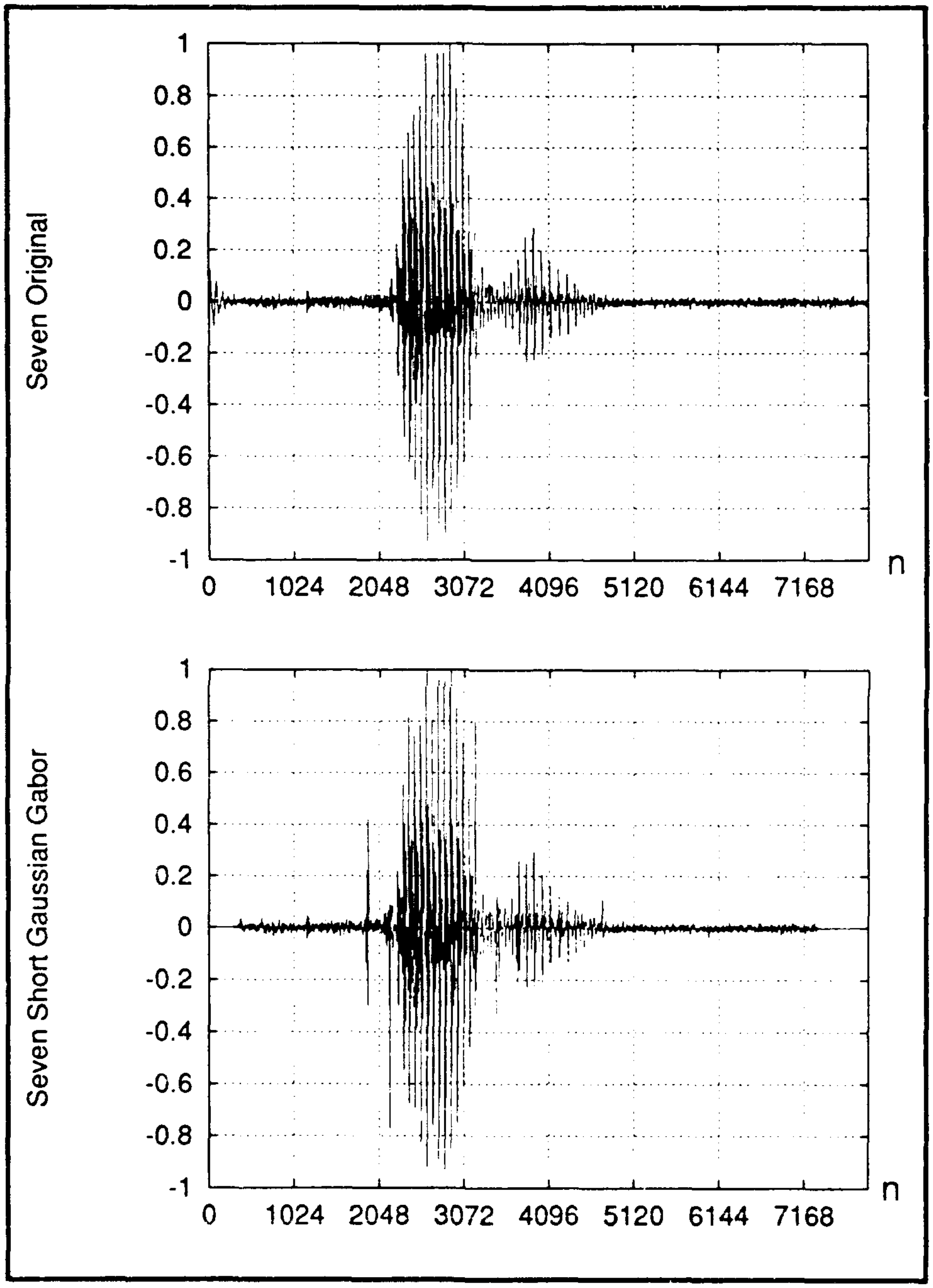

Figure 30. Original Signal and Reconstructed Signal from Short Gaussian Gabor Spectrum

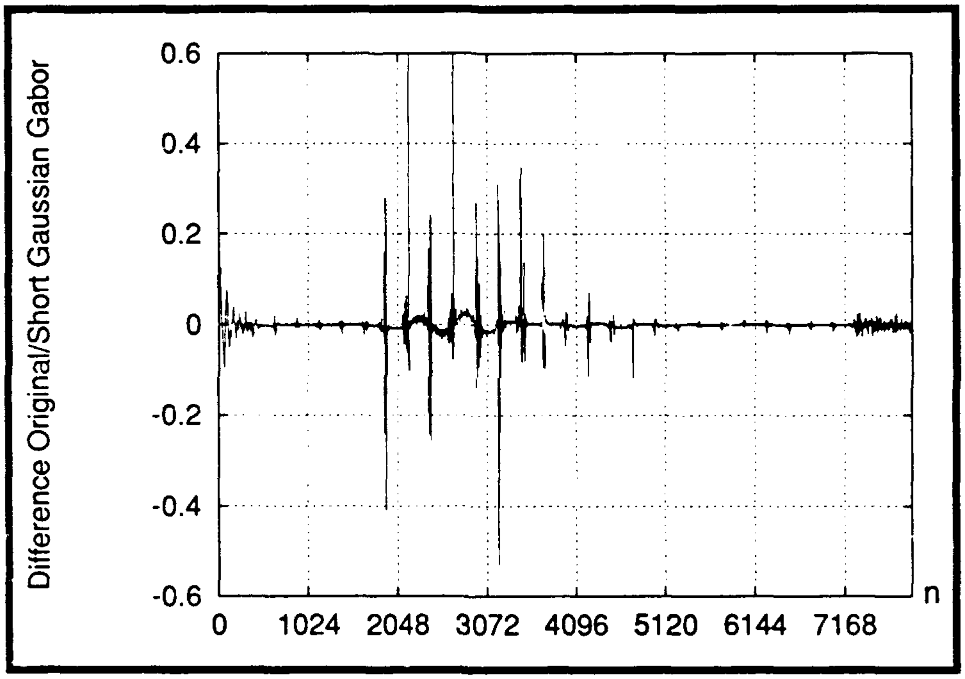

Figure 31. Difference Signal Between Original and Reconstructed from Short Gaussian Gabor Spectrum

$$f_0 = 11$$
$$\tau = 1$$

The units are arbitrary. This signal is sampled at $1/T = 64$ and $N = 64$. The three dimensional plot illustrates the Gabor decomposition of the signal and shows the resulting impulse centered on the time axis at $m = 3$, and centered on the frequency axis at $k = 11$—corresponding exactly with t_0 and f_0 (the indexes in the Gabor lattice plots are off by one due to the plotting software). The bottom left hand graph shows perfect reconstruction as expected.

If, however, the signal is not shifted and modulated at precise integer multiples of time and frequency, respectively, the resulting Gabor representation is quite different as shown in Figure 35. The original signal is shifted and modulated by $t_0 = 3.3$ and $f_0 = 11.2$ in this case. The Gabor lattice now shows *ringing* in both the time and frequency axes. This ringing is due to the uncertainty produced in trying to pinpoint the signal in time and frequency. Indeed, the Gabor representation can be thought as a two-dimensional probability density function. From the Gabor coefficients shown plotted in Figure 35, the reconstructed signal displays the uncertainty produced in the expansion of the original signal. Notice that a small error occurs at $n = 192$ ($t_0 = 3$) as well as at subsequent integer multiples of $n = 64$ (t). This same phenomenon is exhibited in the error signal shown in Figure 33. This is to be expected since naturally occurring speech signals are not expected to have temporal, frequency, and envelope characteristics that perfectly match the window function.

The types of errors shown in Figure 33 are audibly experienced as clicks. These clicks are more pronounced in some words than in others. All hearing test subjects reported the presence of the clicks in the reconstructed signals. However, all subjects reported that these errors were not as obtrusive as the errors experienced in the regenerated signals from the short Gaussian Gabor spectrum. Since audible errors resulted from using the exponential window, no compression/reconstruction tests were performed based on these decompositions. The minimally supported biorthogonal Gaussian function, on the other hand, is important when reconstructing the speech signals from a compressed Gabor representation. This will be examined later in this section.

Table 4 lists the MSEs of all regenerated signals described thus far. Subjectively, the results can be ranked nearly the same as the MSE ranking. The only difference is that the first four entries in the table can be subjectively ranked with equal weight, and each of these are indistinguishable from the original signal.

Table 4. MSE of Reconstructed Signals

Reconstruction Source	MSE
Rectangular Spectrogram	5.3682×10^{-6}
Hamming Spectrogram	5.3682×10^{-6}
Gaussian Spectrogram	5.7554×10^{-5}
Gaussian Gabor	2.7366×10^{-4}
Exponential Gabor	6.9777×10^{-4}
Short Gaussian Gabor	7.9441×10^{-4}

The next series of plots in Figures 36 through 43 show frame 9 ($n = 2304$) of the short-time Fourier spectra and the Gabor spectra for each window used. Frame 9 of the signal corresponds to the first vowel e (ε, from the International Phonetic Alphabet) in the word seven. Figure 36 demonstrates that the spectrum obtained from a Hamming window is nearly the same as the one obtained from the Gaussian window. One would expect that from the similarity of the two window functions. On the other hand the Gabor spectra are quite different, as shown in Figure 37. The Gaussian window in this case produces much more energy below 150 Hz or so. In addition, the Gaussian Gabor decomposition tends to attenuate the energy of the higher formants. Unexpectedly, the one-sided exponential window produces a Gabor spectrum much closer to the short-time Fourier spectrum. Again, these examples are representative of the effects observed in the entire vocabulary.

Figures 38 through 43 show how the LIN eliminates spectral components as a function of competitive bandwidth rule and threshold. The spectral lines shown in these

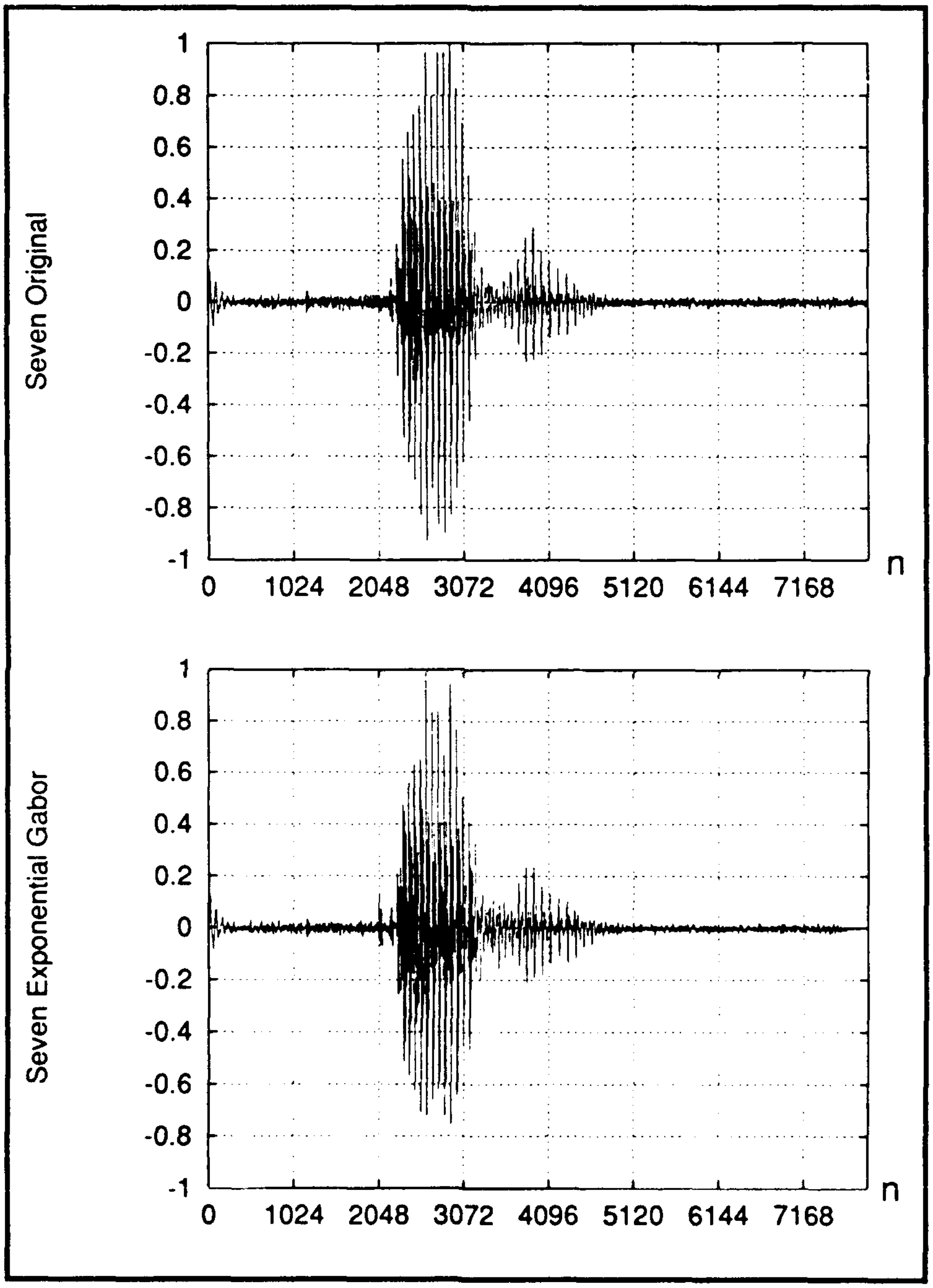

Figure 32. Original Signal and Reconstructed Signal from Gabor Spectrum Using an Exponential Window

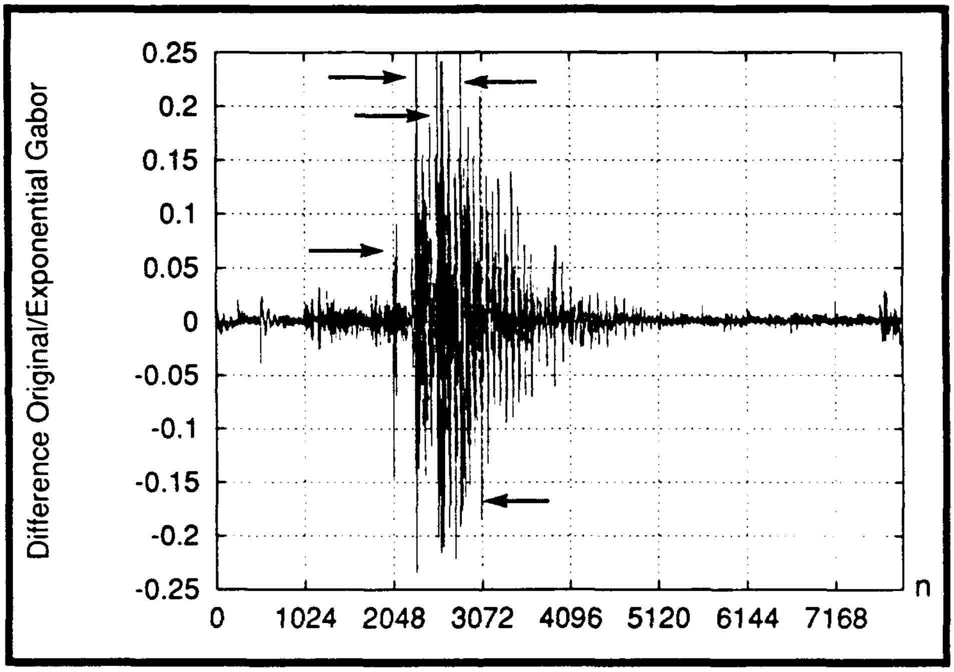

Figure 33. Difference Signal Between Original and Reconstructed from Gabor Spectrum Using an Exponential Window

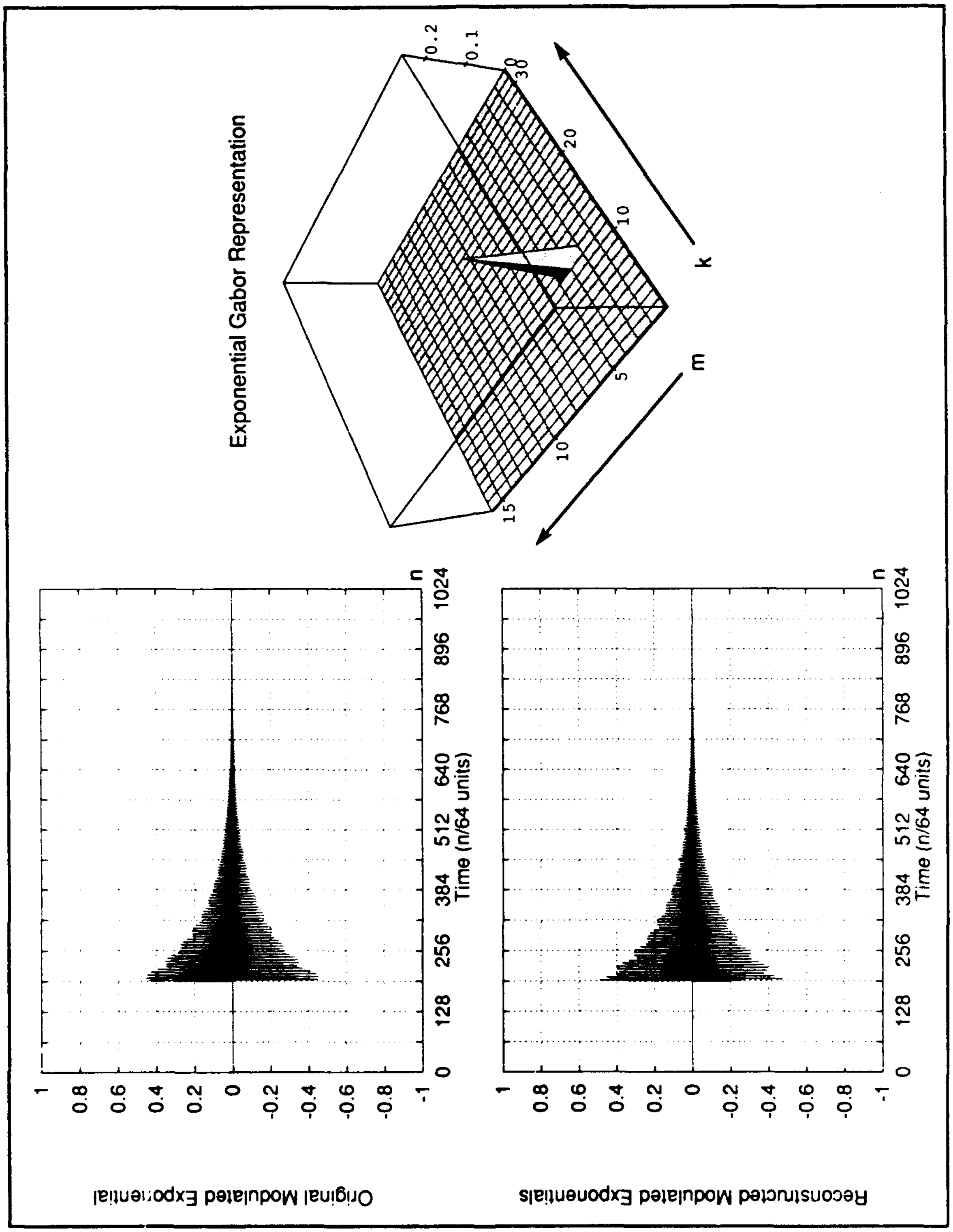

Figure 34. Exponential Gabor Decomposition/Reconstruction Example of Aligned Signal in Time and Frequency

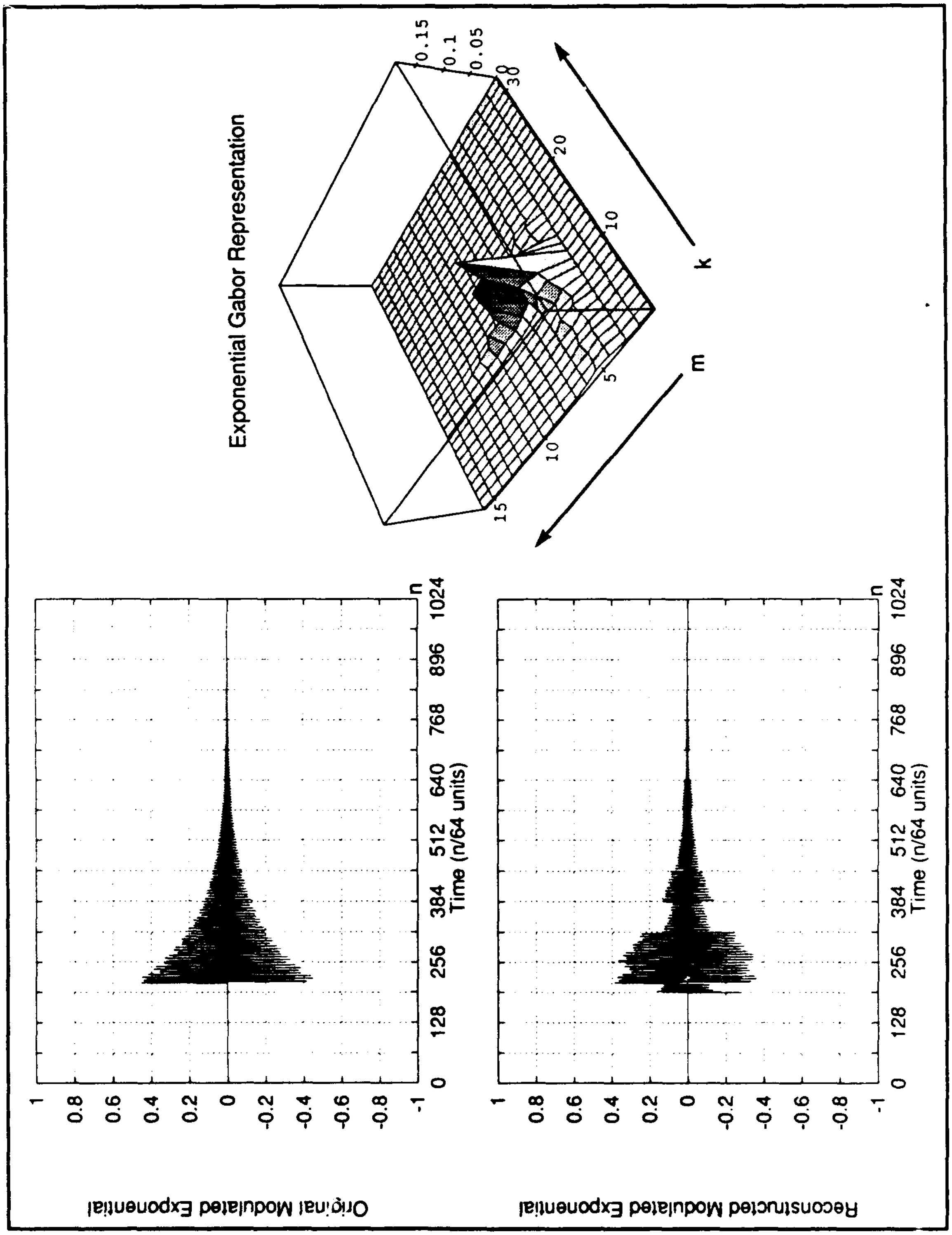

Figure 35. Exponential Gabor Decomposition/Reconstruction Example of Unaligned Signal in Time and Frequency

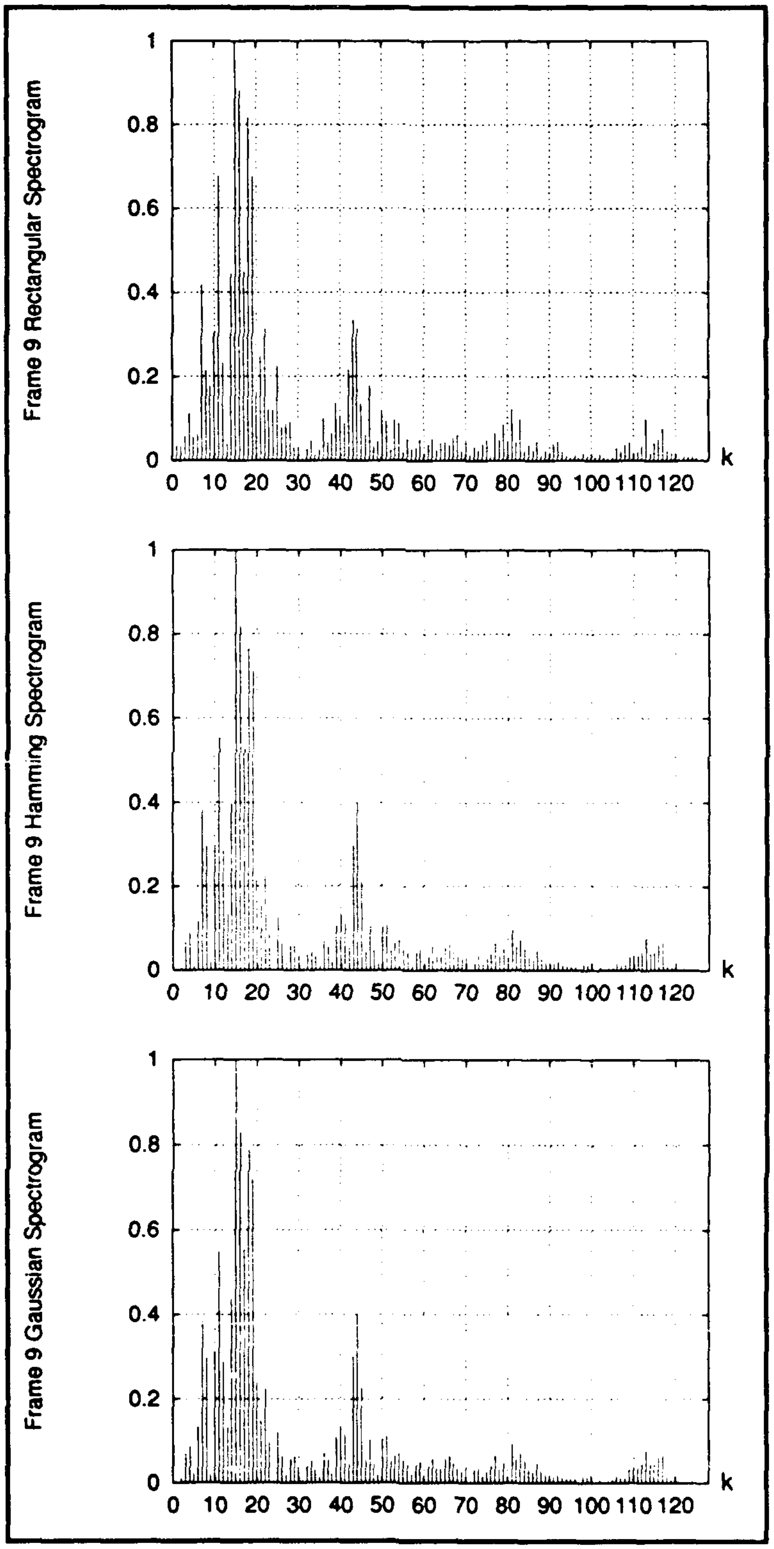

Figure 36. Frame 9 of Spectrogram for Each Window

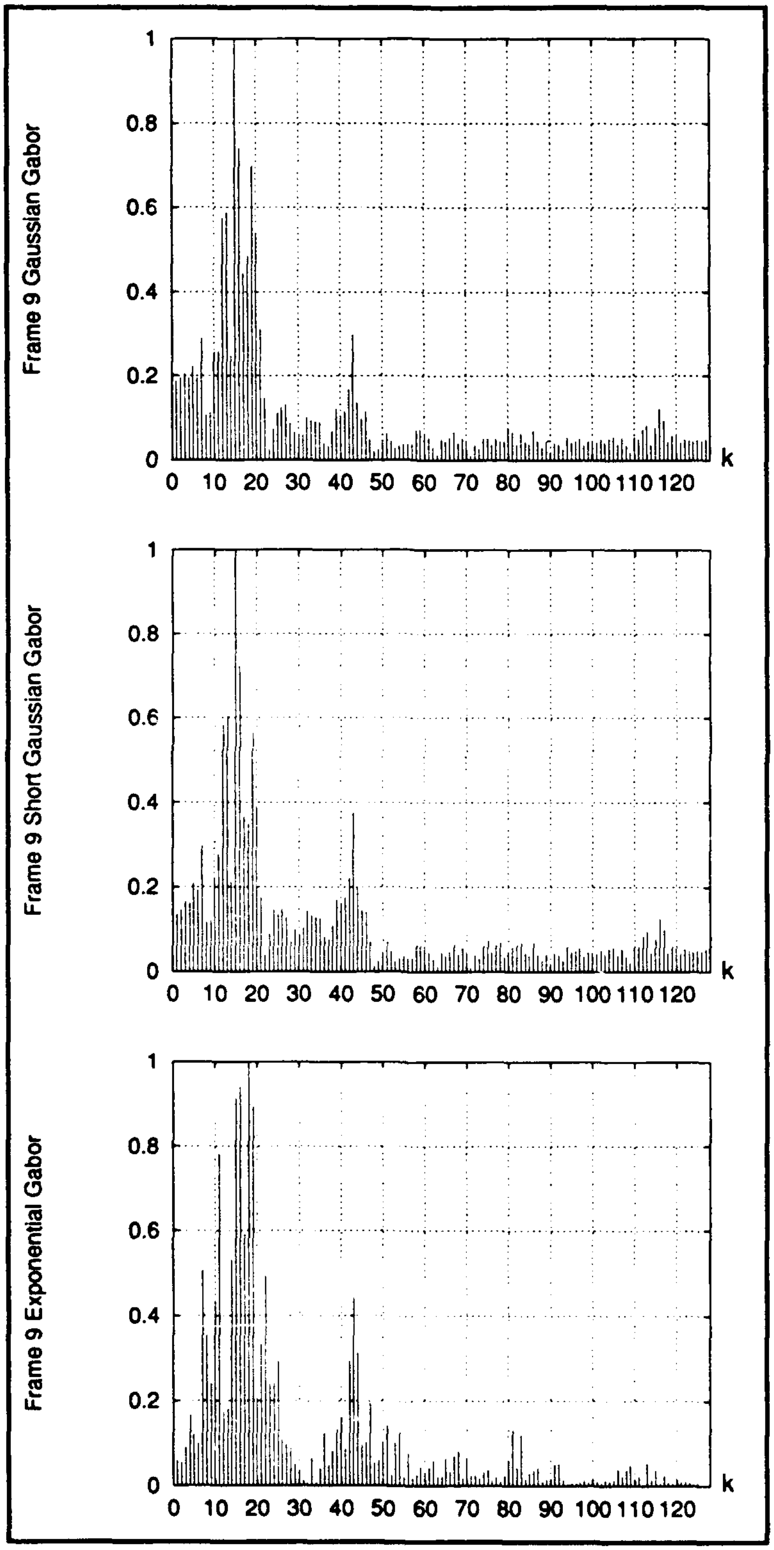

Figure 37. Frame 9 of Gabor Spectrum for Each Window

figures represent the normalized output of the LIN after convergence. One may recall that the competitive bandwidths used to approximate the ERB are more conservative than the CB (see Figure 19). As a result the CB rule produces more compression than the ERB rule. The LIN output function thresholds used were $\theta = .0009, .09$, and are referred to as the low threshold and high threshold in Figures 38 through 43. The high threshold in all compressions tends to eliminate the higher frequency components—above 1500 Hz—due to their low energy. This is evident in Figures 40 and 43. Although it may be difficult to determine from the figures, the LIN consistently finds the local maxima as defined by the competitive bandwidths. These local maxima correspond the the glottal frequency (the first maximum chosen) and its harmonics, including the the formant peaks.

Before describing the results of regenerating the signal from the compressed spectra, the time-aliasing effects described in the previous chapter are shown in Figures 44 and 45 for two cases. Figure 44 shows the plot of the reconstructed and difference signal from the compressed Fourier spectrum using a Hamming window, and, similarly, the plots shown in Figure 45 are derived from the compressed Gaussian Gabor Spectrum. In the former case, the errors can be described as large amplitude errors concentrated around the window edges, whereas in latter case, the errors can be described as spiky noise. The spiky noise in the Gabor case was predicted earlier in the discussion pertaining to Figure 28. Again, the conjecture is that the spikes of the biorthogonal function seem to leak through if the window function does not entirely cancel them out.

The windowing process described in Eq (36) completely eliminates the time-aliasing effects in the case of the short-time Fourier spectra. Consequently, the best results were obtained—both subjectively and in terms of MSE—from these representations. The compressed Gabor spectra did not reproduce signals as well the Fourier spectra, even with windowing. However, all signal reproductions revealed that a truncated biorthogonal function should be used when compressing the Gabor spectrum. This is illustrated in Figures (46) and (47). The spike noise cannot be fully eliminated when a maximally supported biorthogonal function is used. With a truncated biorthogonal function, spikes

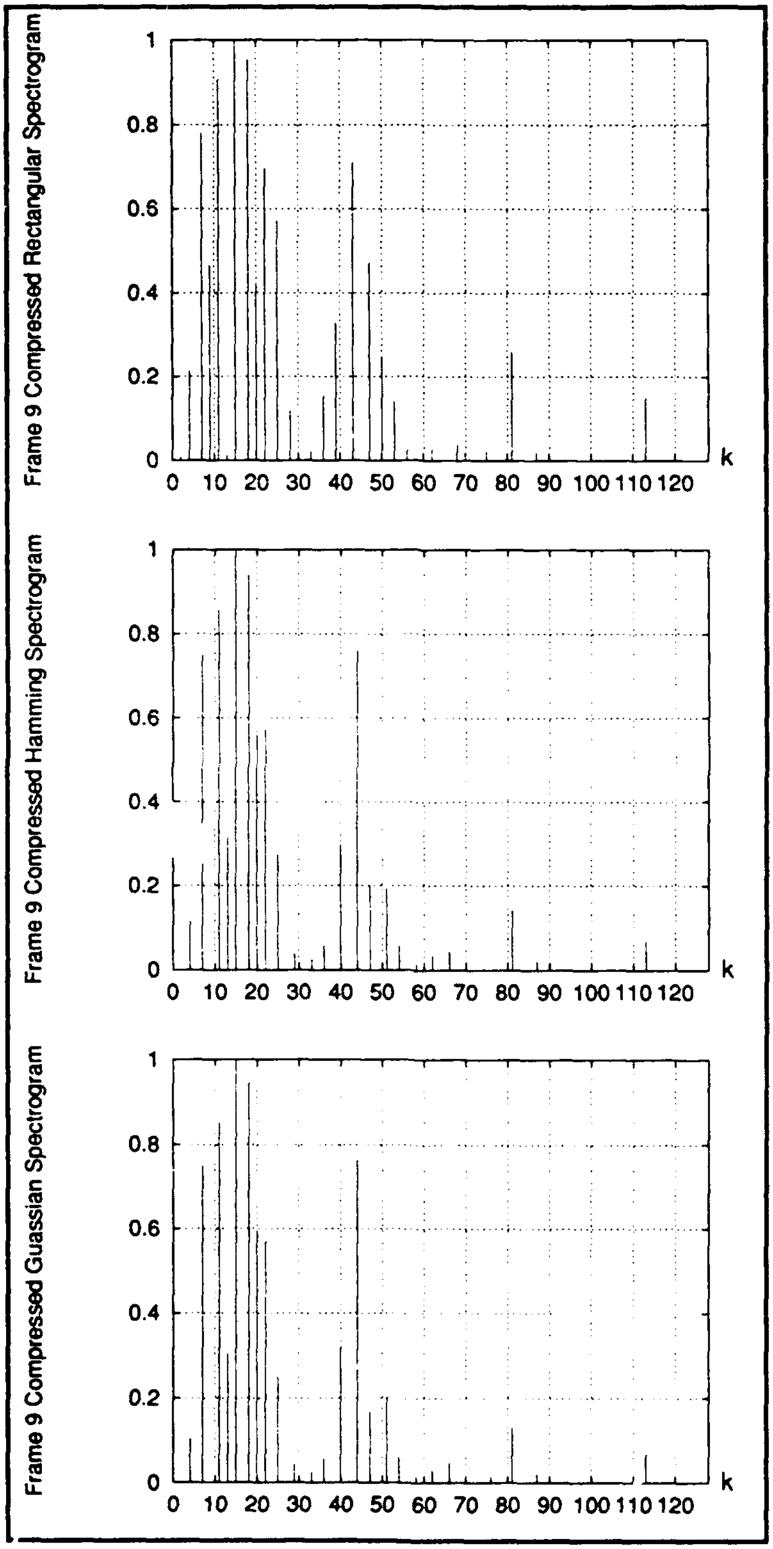

Figure 38. Compression of Spectrogram Using ERB Rule and Low Threshold

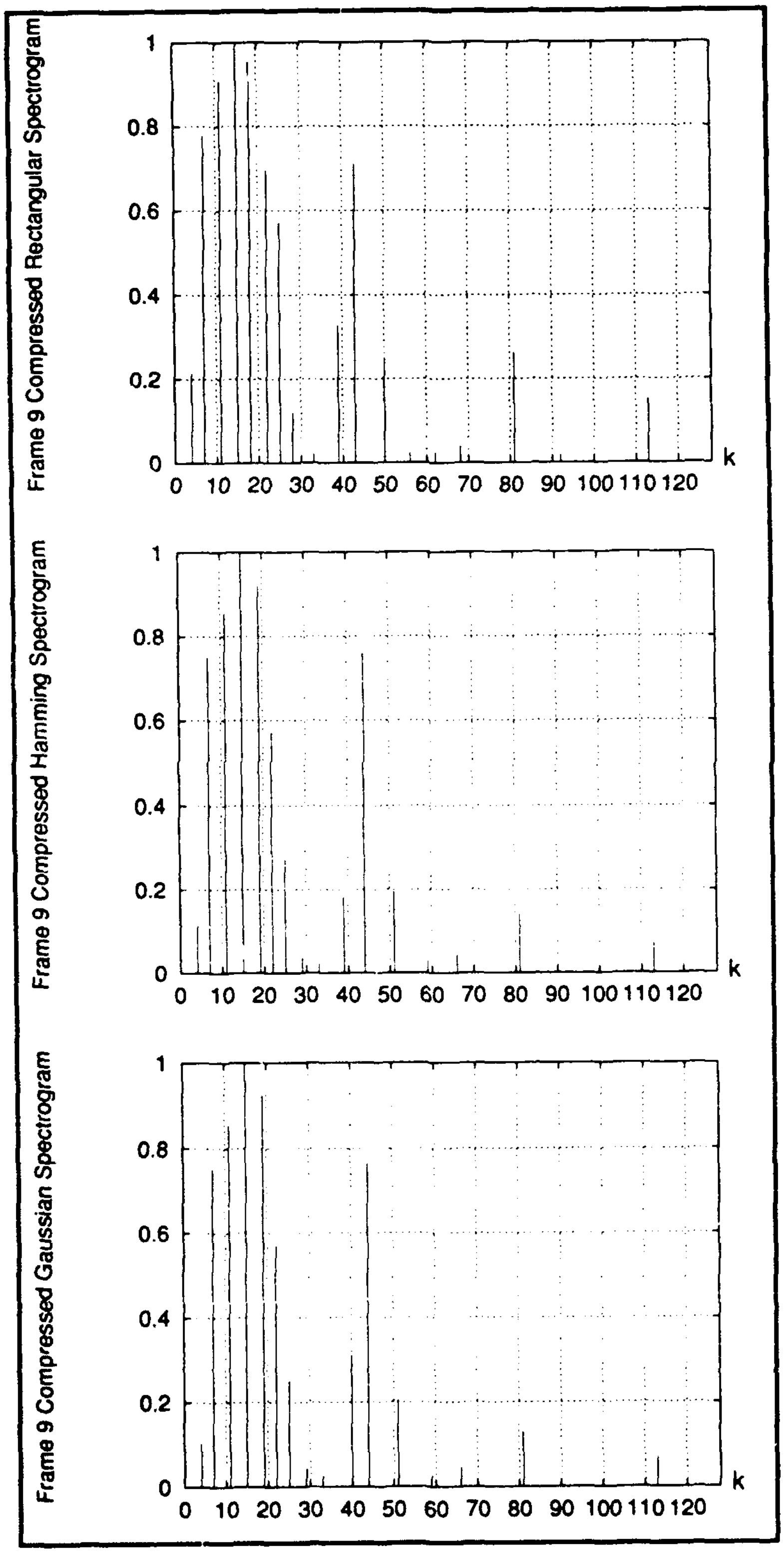

Figure 39. Compression of Spectrogram Using CB Rule and Low Threshold

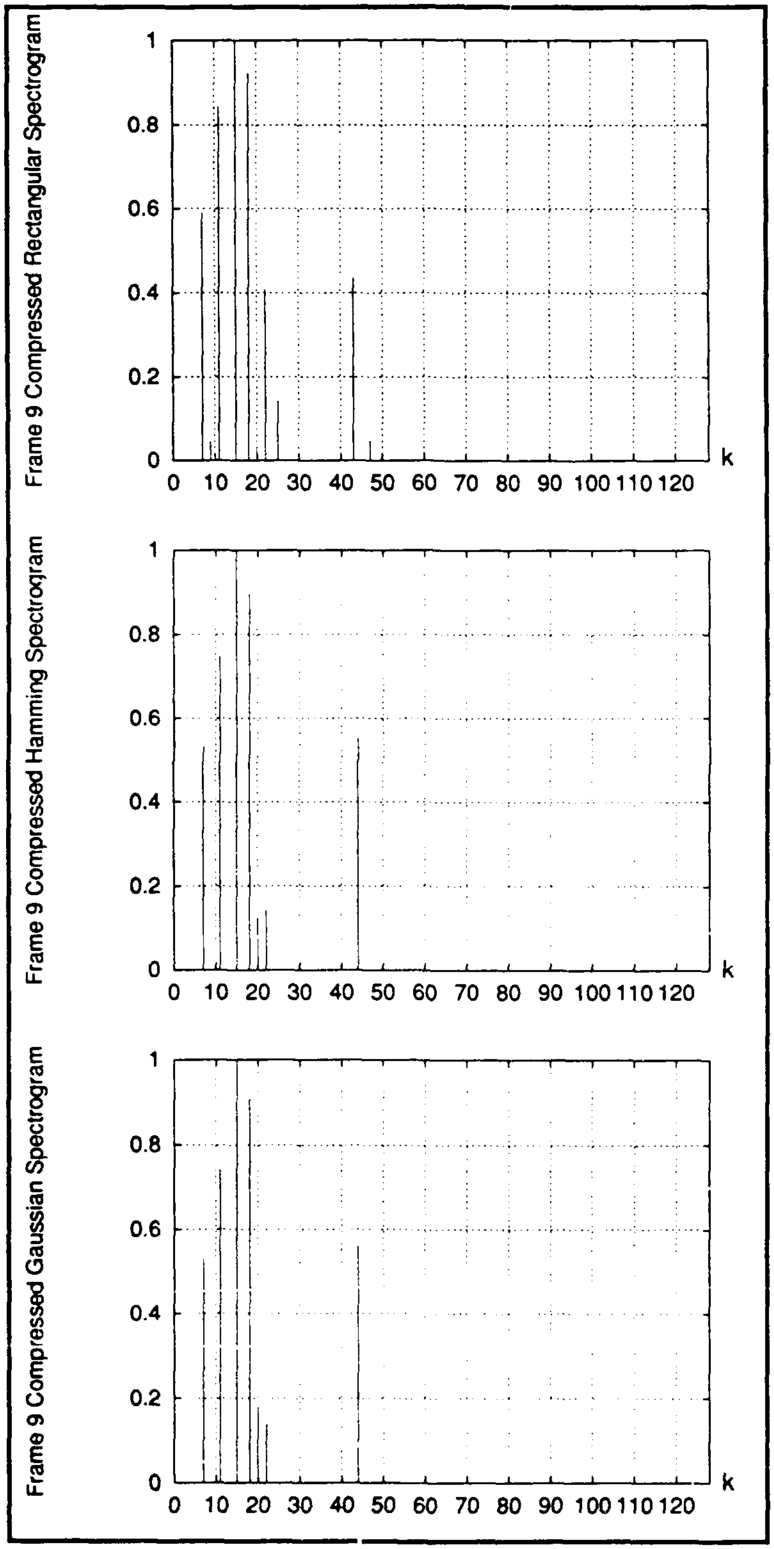

Figure 40. Compression of Spectrogram Using CB Rule and High Threshold

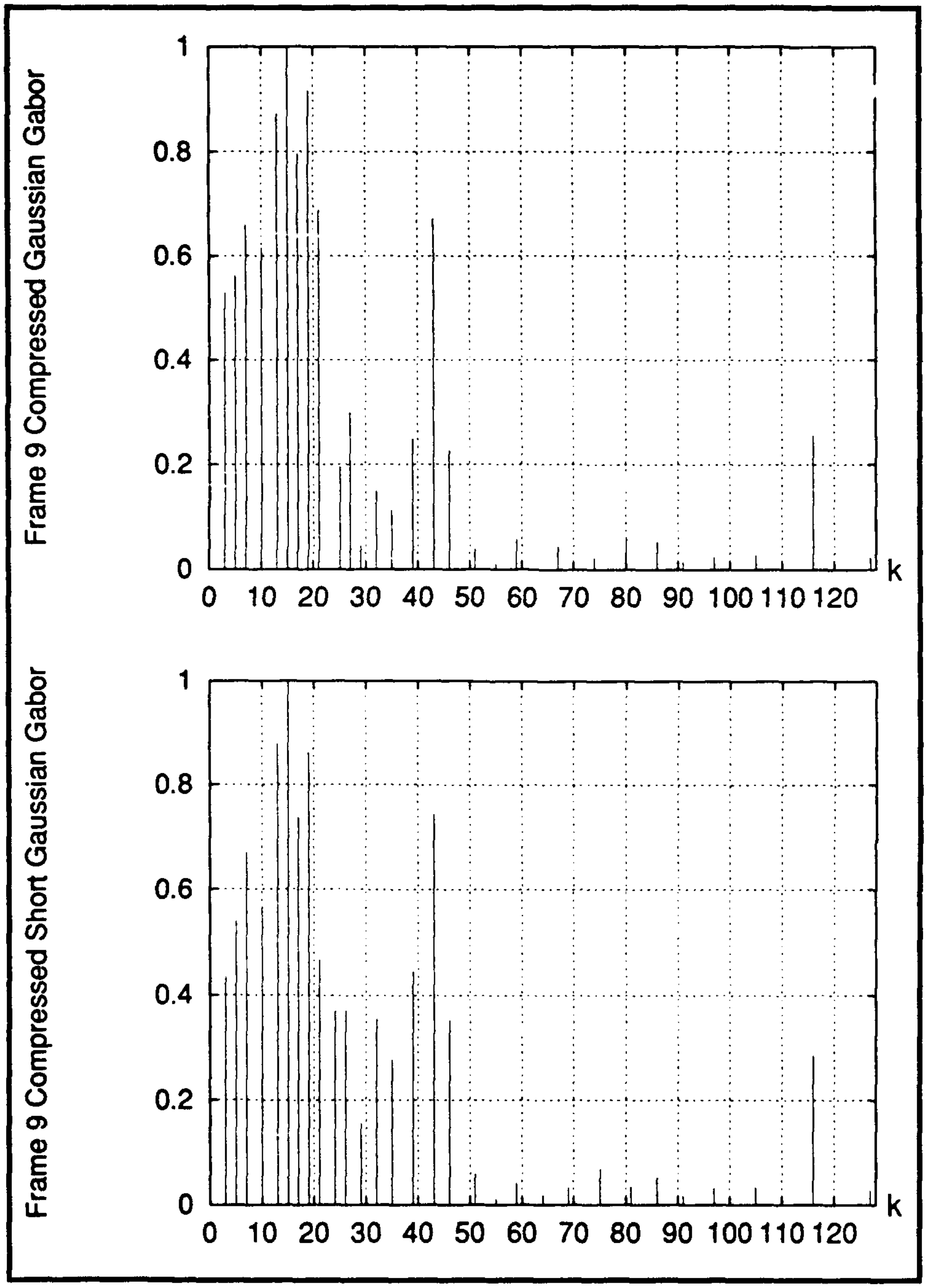

Figure 41. Compression of Gabor Spectrum Using ERB Rule and Low Threshold

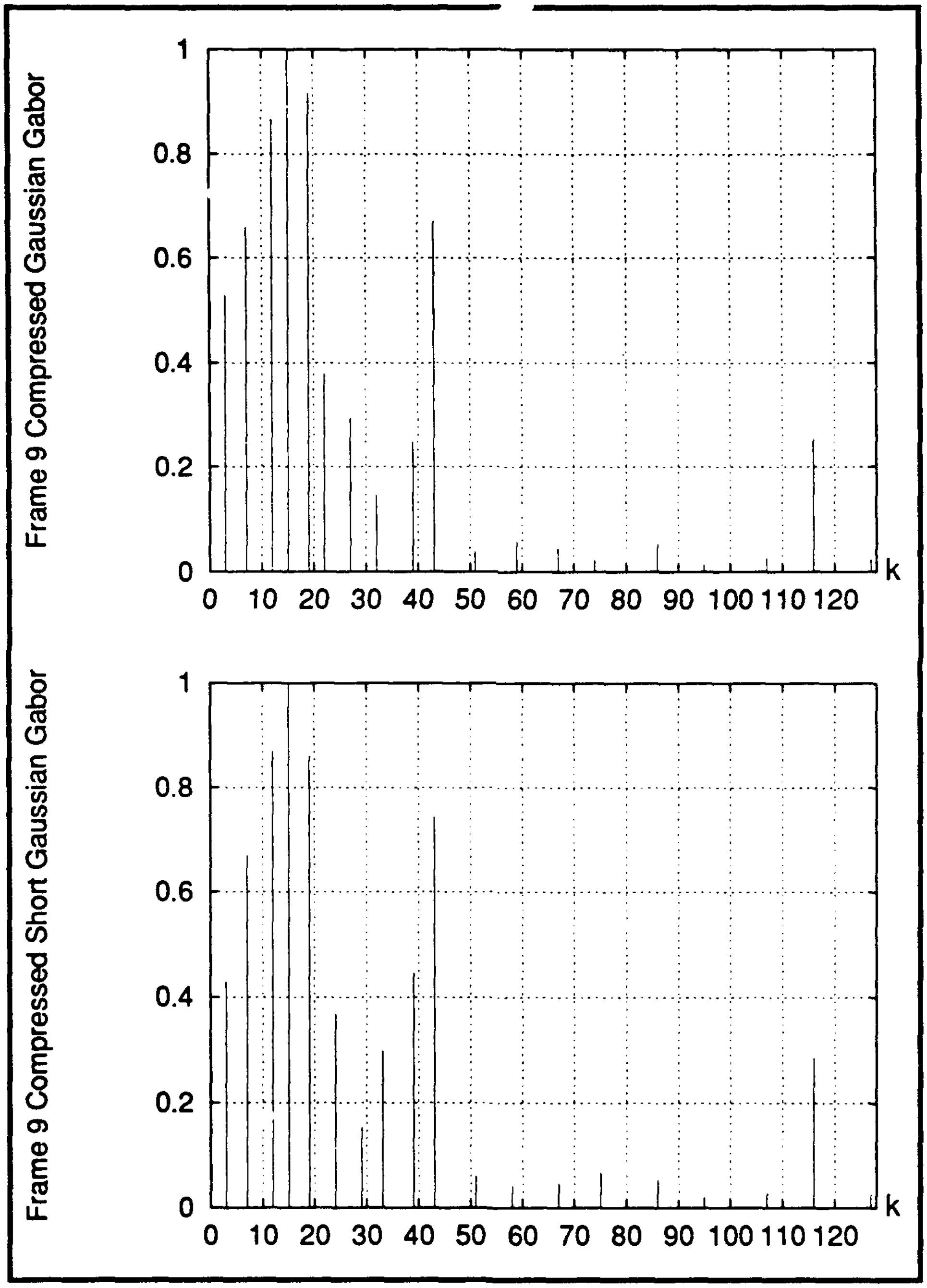

Figure 42. Compression of Gabor Spectrum Using CB Rule and Low Threshold

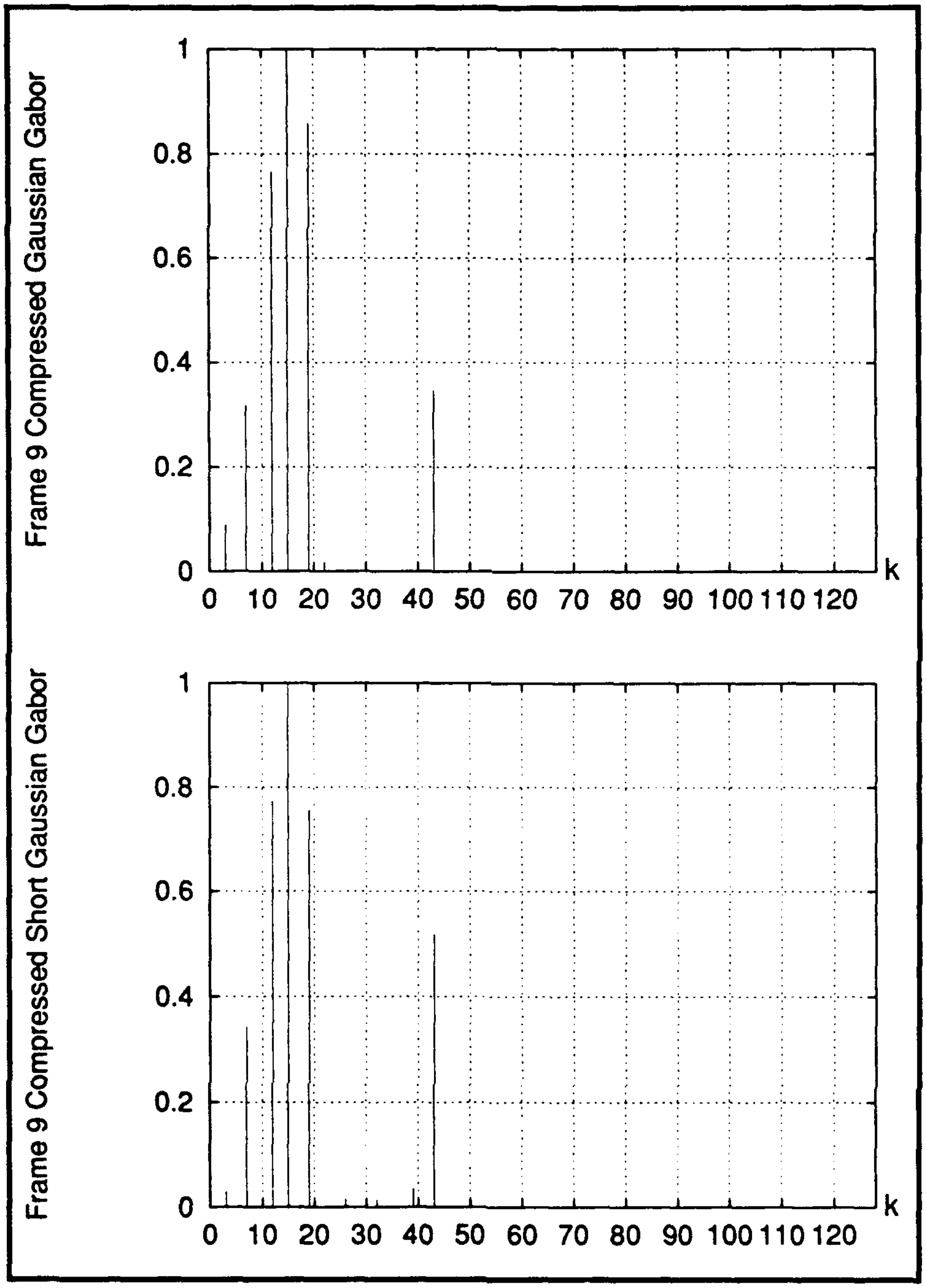

Figure 43. Compression of Gabor Spectrum Using CB Rule and High Threshold

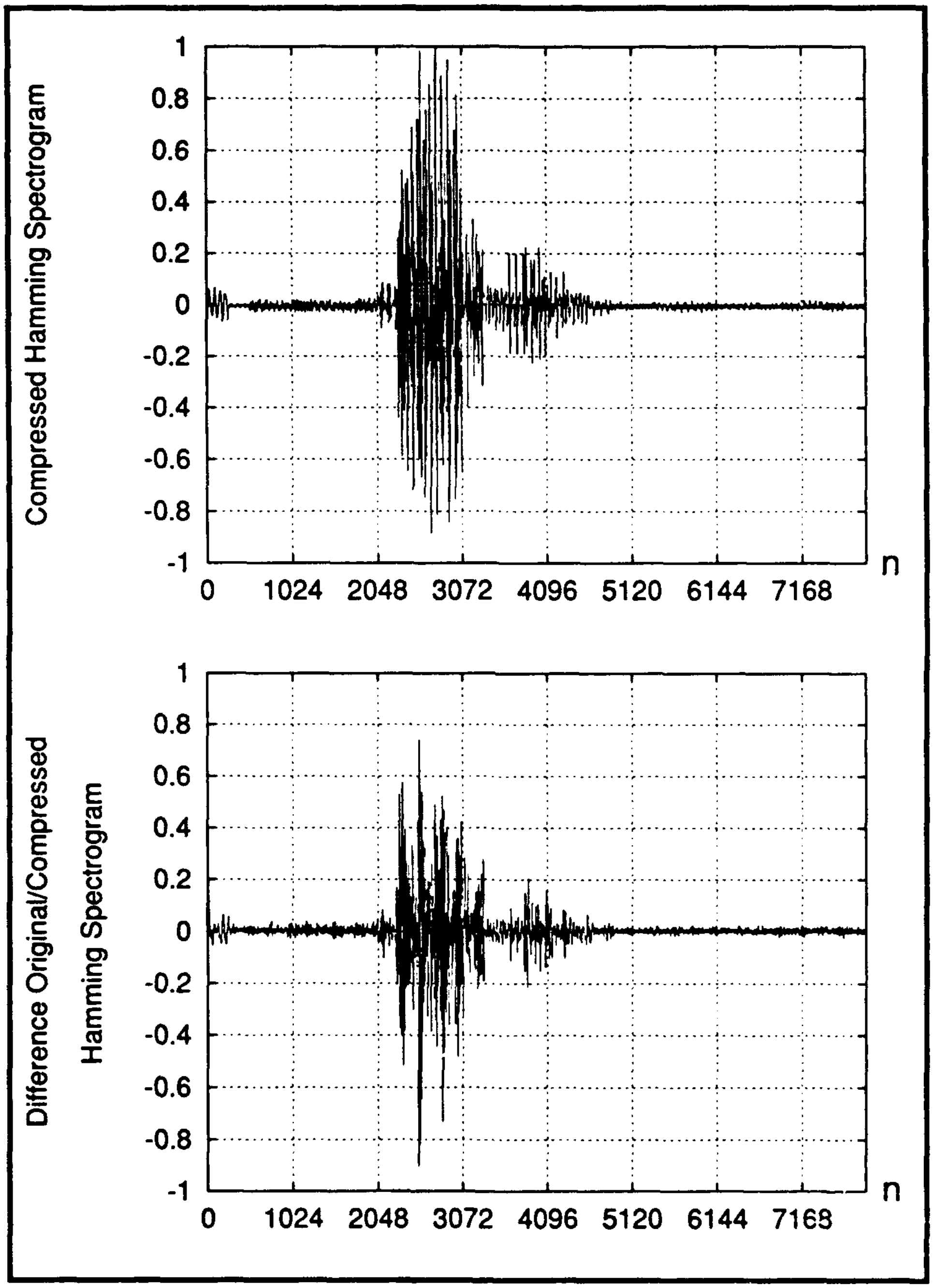

Figure 44. Aliasing Effects Obtained from Compressed Spectrogram Using a Hamming Window

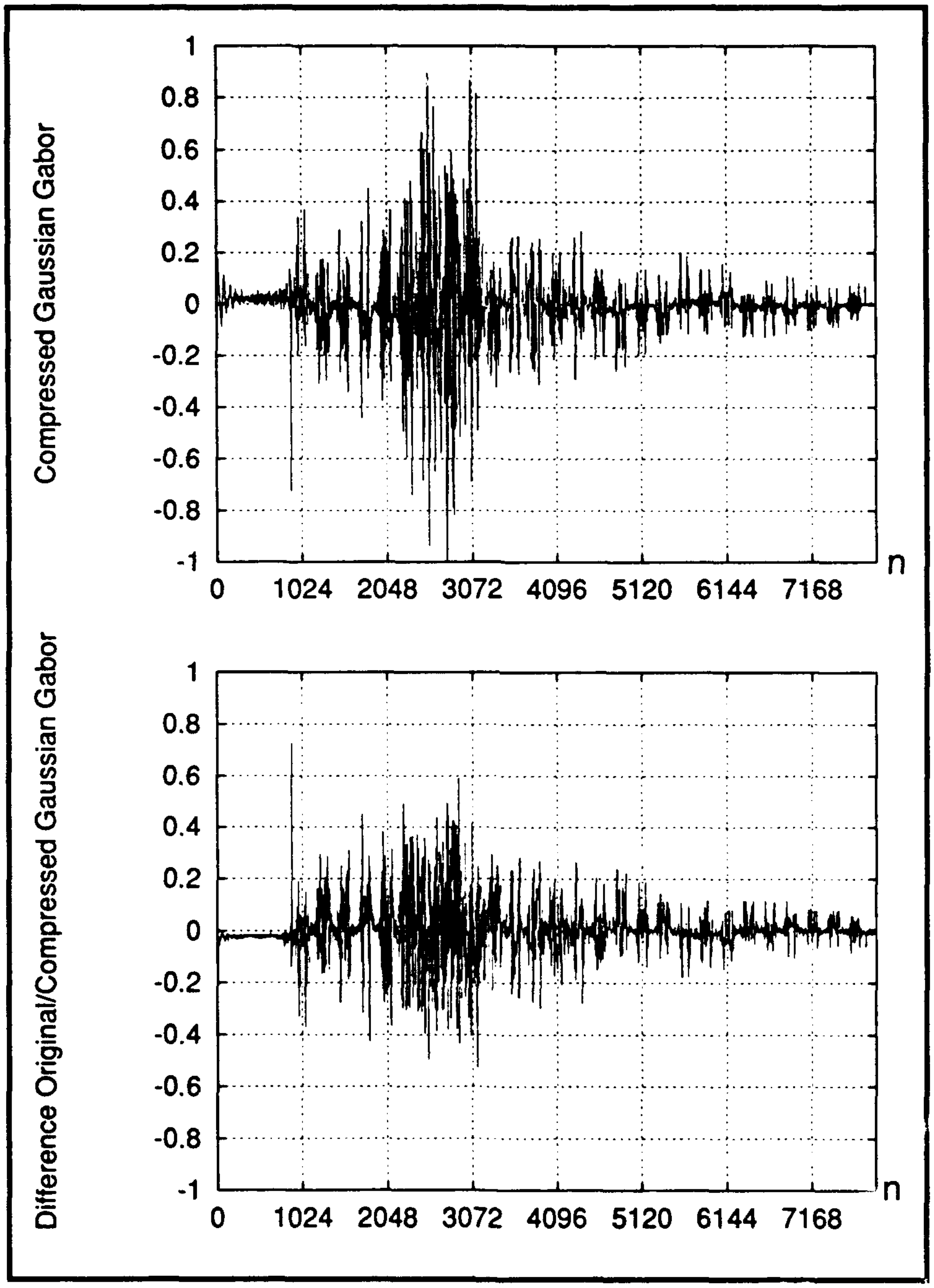

Figure 45. Aliasing Effects Obtained from Compressed Gabor Spectrum Using a Gaussian Window

do not appear, however, the quality of the reconstructed speech can be characterized as rough sounding.

As previously shown, the amount of compression can be controlled by either increasing the competitive bandwidths or increasing the output threshold of the LIN. The reconstructed speech obtained from the compressed spectra using a LIN tuned to the ERB rule was virtually indistinguishable (subjectively) from the speech obtained from the compressed spectra using the CB rule. This is true only for a low output threshold. Increasing the LIN output threshold tends to degrade the fricatives. However, this is preferable to increasing the competitive bandwidths much beyond the CB rule. Speech compressed using the LIN tuned to twice the CB rule produced markedly deteriorated reconstructed speech. This result supports the argument that CB rule better describes the filter bandwidth characteristics of the cochlea than the ERB rule.

The best results, overall, were produced from the short-time Fourier spectrum using a compactly supported Gaussian window. Also, unexpectedly, better compression resulted from this representation than any other representation compressed with an identically tuned LIN. Table 5 lists the compression ratios obtained using a LIN tuned to the CB and twice the CB rule with two values of output threshold for each window used. The compression ratio is defined as

$$R_c = N_t / N_c \tag{76}$$

where N_t is the total number of coefficients, and N_c are the total number of coefficients left after compression.

Speech compressed using the LIN tuned to the CB rule produced clearly intelligible speech when resynthesized, even when a high LIN output threshold was used. Table 6 lists the MSEs obtained from the respective compressed spectra using the CB rule and an output threshold $\theta = .09$. As before, the subjective ranking coincides with the MSE ranking.

Although the reconstructed speech from the compressed short-time Fourier spectrum

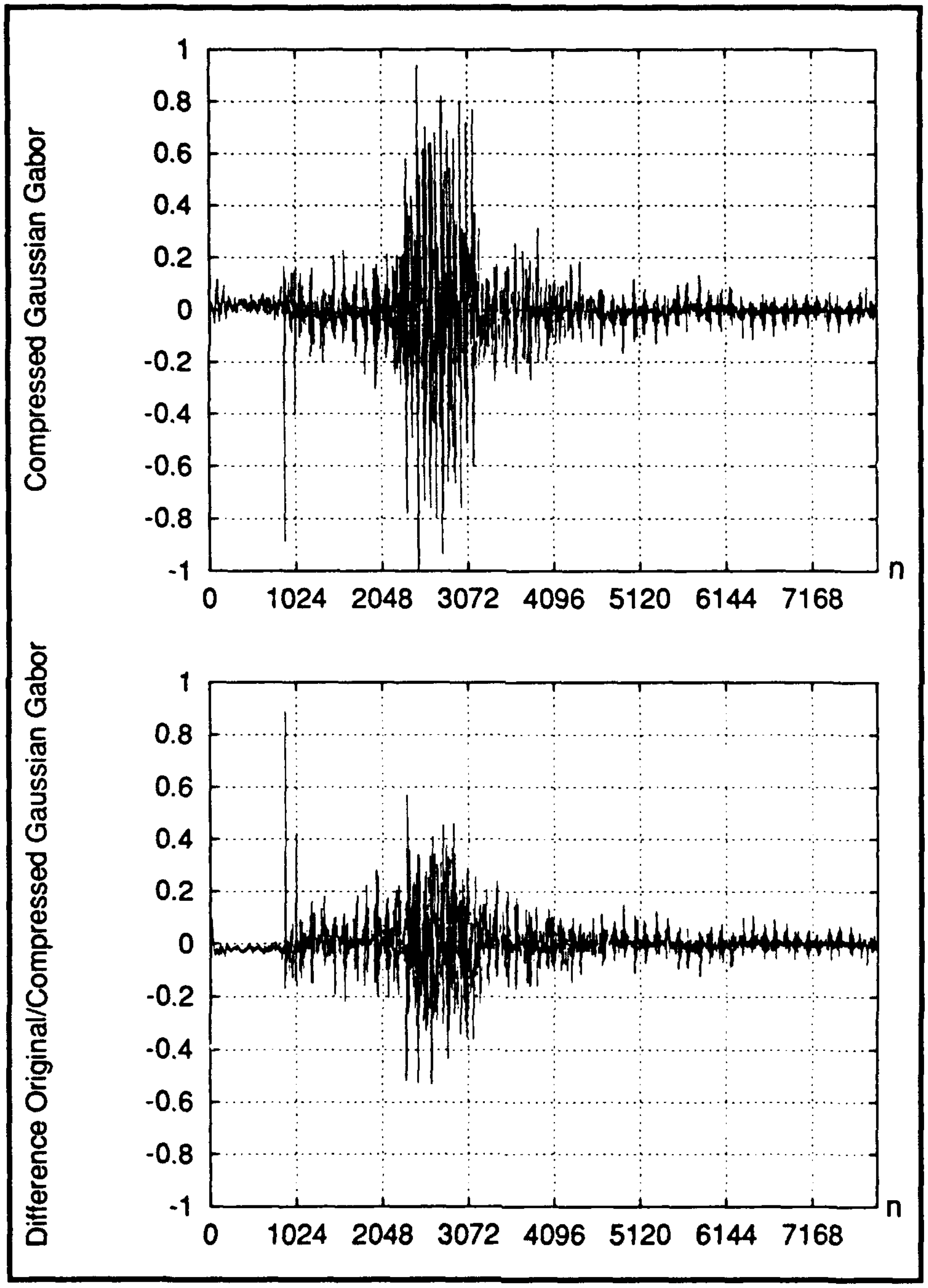

Figure 46. Reconstructed and Difference Signal from Compressed Gabor Spectrum Using a Gaussian Window

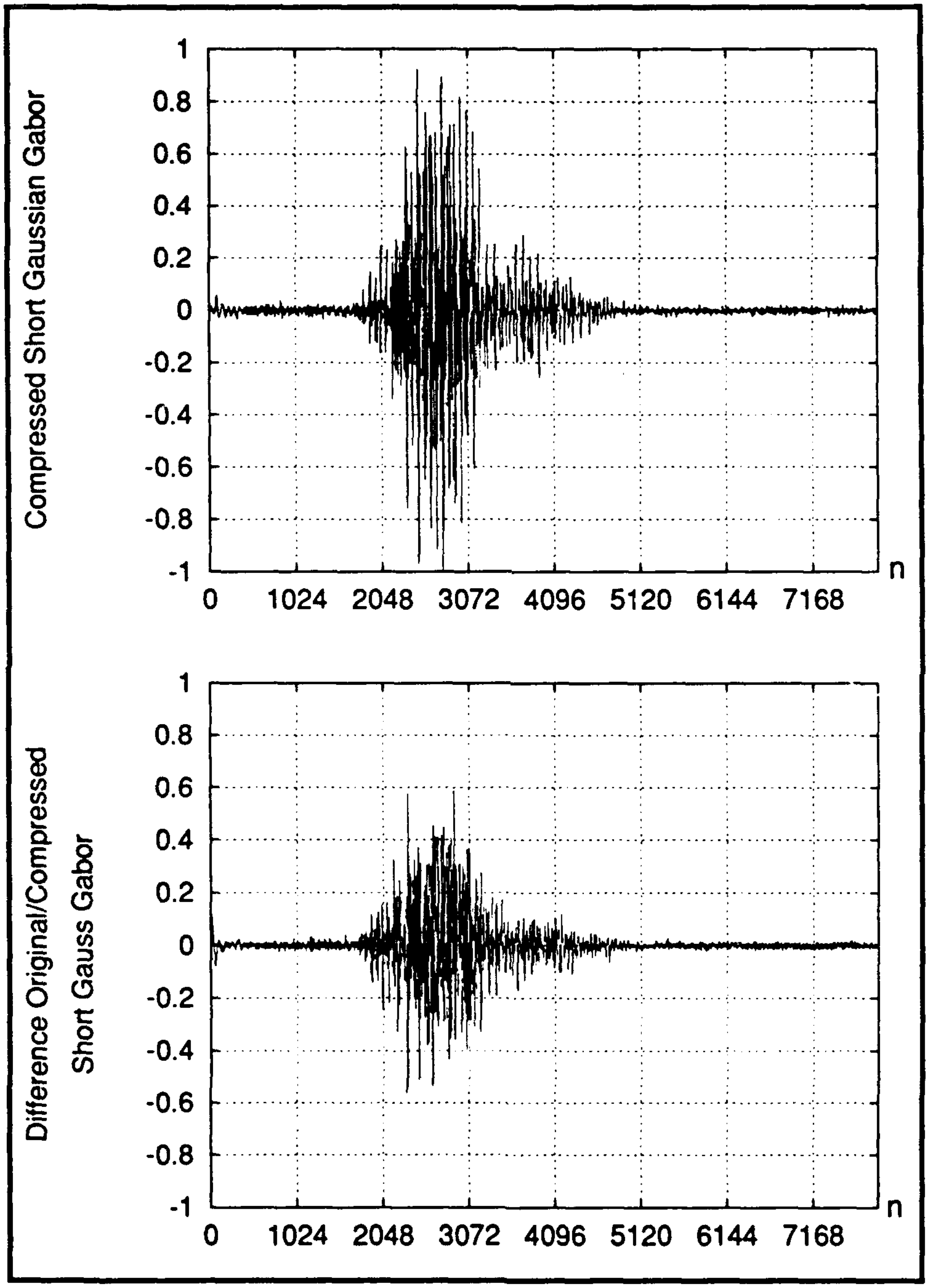

Figure 47. Reconstructed and Difference Signal from Compressed Gabor Spectrum Using a Gaussian Window with Truncated Biorthogonal Function

Table 5. Compression Ratios Obtained from Short-Time Fourier Spectrum

Compression Ratios (R_c)				
LIN Tuning		Window		
Rule	Threshold	Rectangular	Hamming	Gaussian
CB	.0009	8.84	11.23	11.77
CB	.0900	18.08	23.55	24.72
2×CB	.0009	14.95	18.54	19.40

Table 6. MSE of Reconstructed Signals from Compressed Spectra

Reconstruction Source	MSE
Gaussian Spectrogram	1.8736×10^{-3}
Hamming Spectrogram	1.9066×10^{-3}
Rectangular Spectrogram	3.2633×10^{-3}
Short Gaussian Gabor	4.3132×10^{-3}
Gaussian Gabor	4.8427×10^{-3}

was intelligible, it can not be claimed that this speech is of toll quality. In tests performed on continuous speech, musical quality tones are perceived around some fricatives in the reconstructed speech. The phrase processed in these tests was presented in Chapter I, Section *Scope*.

This section concludes with an estimate of the bit rate achieved by compressing the Gaussian window derived Fourier spectrum of the test phrase. No source or channel coding is assumed in the bit rate estimate. The LIN was tuned with the CB rule and an output threshold of .09. A compression ratio of 26.81 was achieved in this case. This translates to an average of 4.774 coefficients per frame. Since the frames are overlapped by 50%, an average of approximately 10 coefficients per 31.95 msec was sufficient to represent the signal. Approximately 31 frames are needed for each second of speech. In order to find the bit rate, the total number of bits that represent the frequency index, magnitude, and phase of each coefficient is established. The total number of bits needed for the frequency index is fixed at 7. The number of bits used to represent the magnitude and phase were decreased to 4 and 2 respectively; down from 16 bits each. This reduction in the number of bits did not significantly degrade the speech signals (neither in MSE nor subjectively). Thus, the total number of bits necessary to represent each coefficient is 13, times 10 coefficients per frame, times 31 frames per second yields 4030 bits per sec. This compares favorably with the 4800 bit rate speech demonstrated by McMillan [49].

Results Based on Affine Wavelet Spectra

As in the previous section, the resynthesized signal from all the coefficients of the affine wavelet spectrum—using the Haar and the Morlet wavelets—are first examined. Afterwards, the result of compressing the affine wavelet spectrum via the LIN, as defined in the previous chapter, is presented.

The regenerated signal from the Haar wavelet obtained coefficients is nearly identical to the original signal, as shown in Figure 48. The error obtained from this reconstruction is almost the same as the error obtained from the signal resynthesized from the short-time Fourier spectrum using the rectangular or Hamming windows. The high fidelity of this reconstructed signal is to be expected due to the completeness of the Haar wavelet set.

In the previous chapter, it was mentioned that four times as many Morlet wavelet levels than are used in this effort are necessary for a complete representation. As a result, a large error between the original and the reconstructed signal from the Morlet wavelet decomposition is to be expected. Figure 49 verifies this. However, what was not expected is that the subjective quality of this reconstructed signal is quite good, although *different* from the original and somewhat noisier.

The Gaussian envelope of the Morlet wavelet–$\exp[-(n/N)^2/2]$, as presented in Eq (61)—has a variance $\sigma^2 = 1$ if one defines the general form of the Gaussian as $\exp[-t^2/(2\sigma^2)]$. In this form, the Morlet wavelet produces quite a large overlap during the computation of the wavelet transform. The variance was reduced to $\sigma^2 = 1/(2\pi)$—in order to reduce the overlap—and the signals were decomposed and resynthesized with the new wavelet set. The result, represented in Figure 50, was a reduction in the MSE between the original and reconstructed signals. The listening test subjects reported that these resynthesized signals sounded *closer* to the original than the signals resynthesized from the wider variance Morlet wavelet decompositions. Table 7 lists the MSEs of the three regenerated signals described in this section.

Table 7. MSE of Reconstructed Signals From Wavelet Coefficients

Reconstruction Source	MSE
Haar Spectrum	5.3224×10^{-6}
Morlet Spectrum	3.2461×10^{-3}
Narrow σ^2 Morlet Spectrum	9.8497×10^{-4}

The reconstructed speech from the compressed affine spectrum—by the method described in Chapter II—was intelligible but of poor quality, regardless of the wavelet used. This reconstructed speech has a very rough quality associated with it. Even when reducing the compression by allowing only nearest neighbor competition in the LIN, the resultant reconstructed speech is rough and muffled. Figures 51 through 53 show the result of compressing all eight Haar generated coefficient levels of frame 9 of the speech signal using the CB rule. At each level, the coefficients have been duplicated an appropriate number in order to span 128 points. For example, at level 0, the only coefficient found at that level is duplicated 128 times, at level 1, both coefficients found at that level are duplicated 64 times, and so on. This provides a convenient method of illustrating the affine spectrum. Recall that the 128 points are associated with a 31.95 msec. interval. The result of compressing the coefficients are shown as shaded areas in the plots. Figures 54 through 55 show the result of compressing the same frame with only nearest neighbor competition. Clearly, this method of compressing the affine spectrum is not useful for regenerating good quality speech.

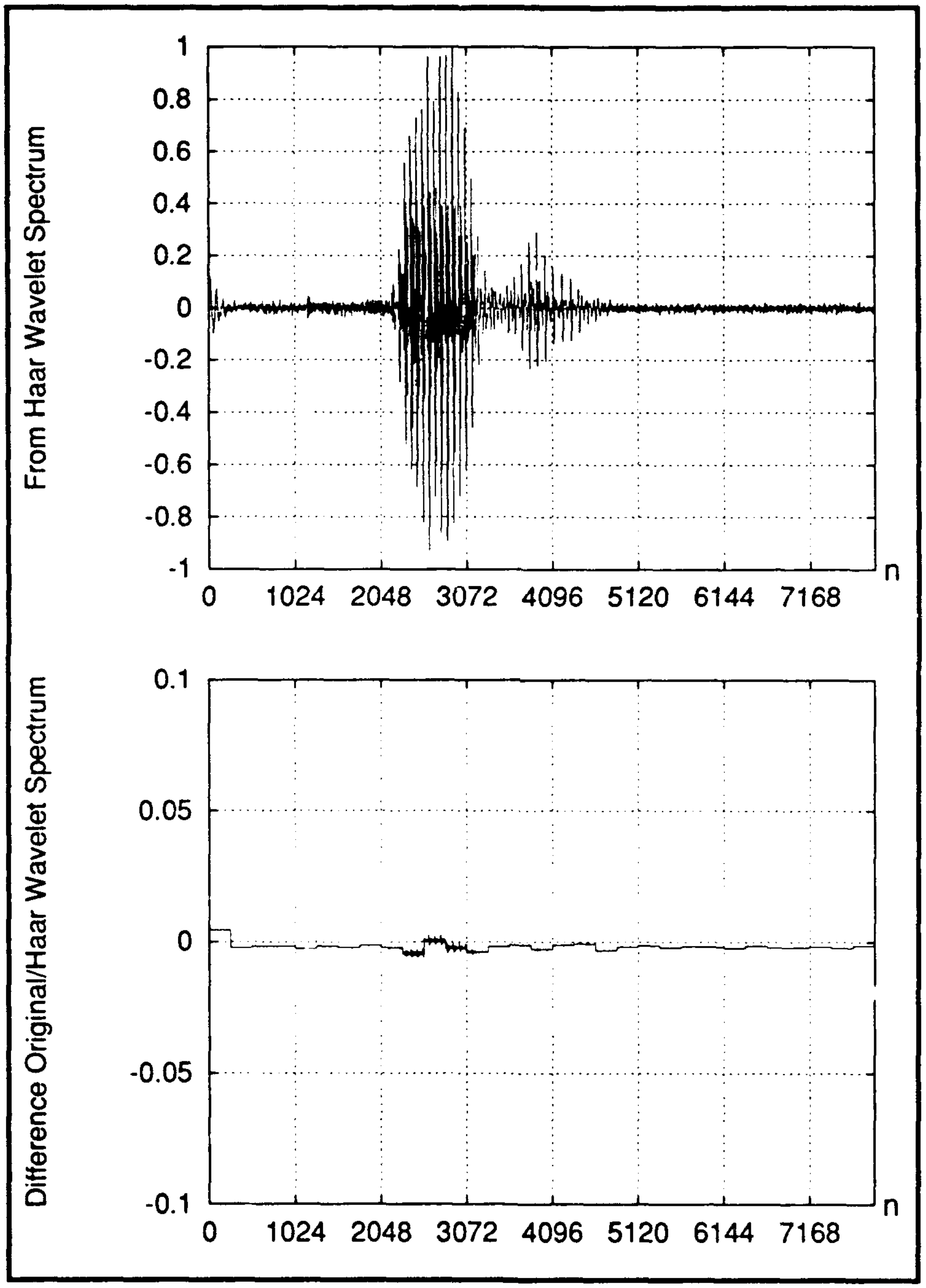

Figure 48. Reconstructed and Difference Signal from Haar Wavelet Coefficients

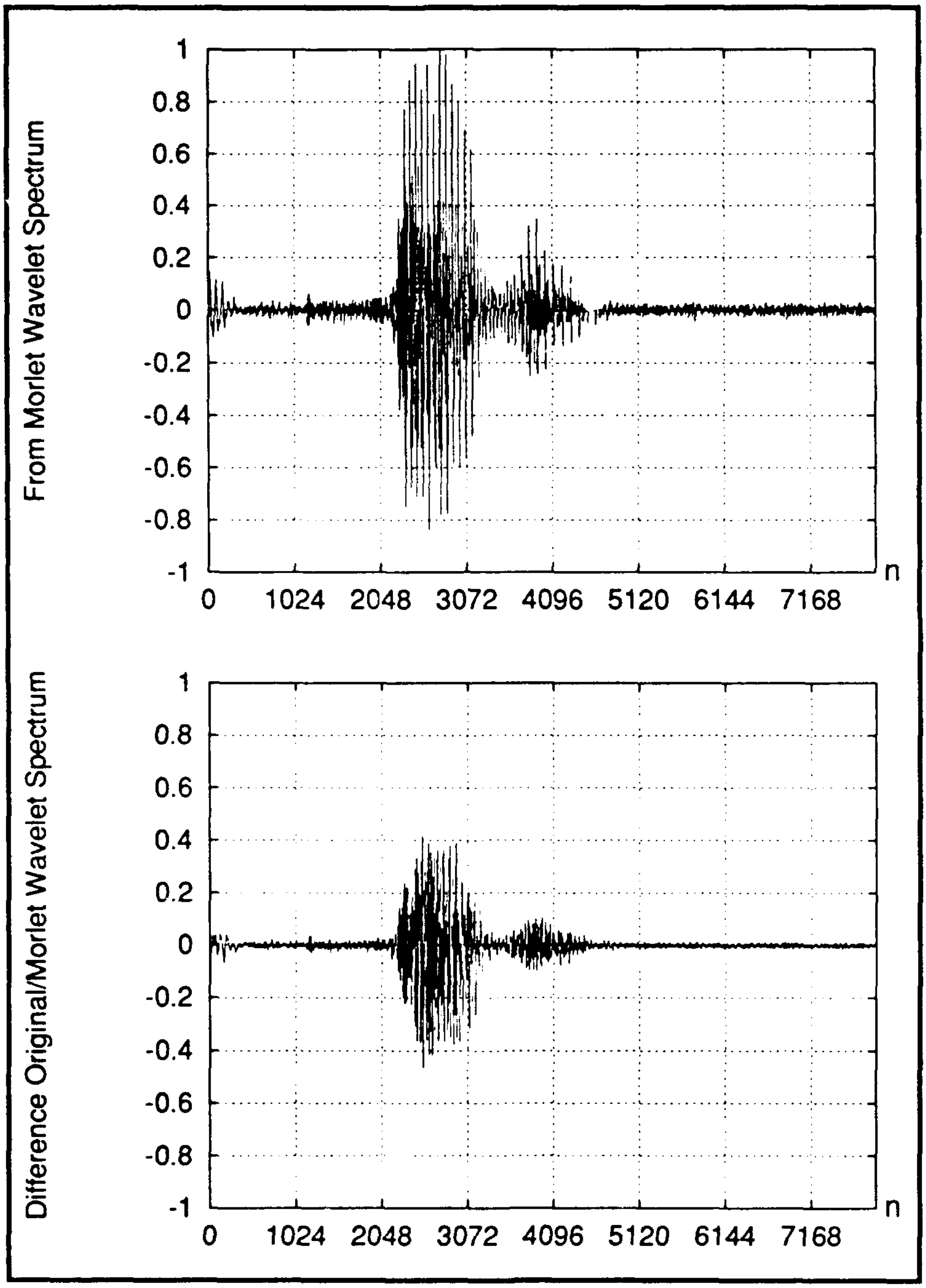

Figure 49. Reconstructed and Difference Signal from Morlet Wavelet Coefficients

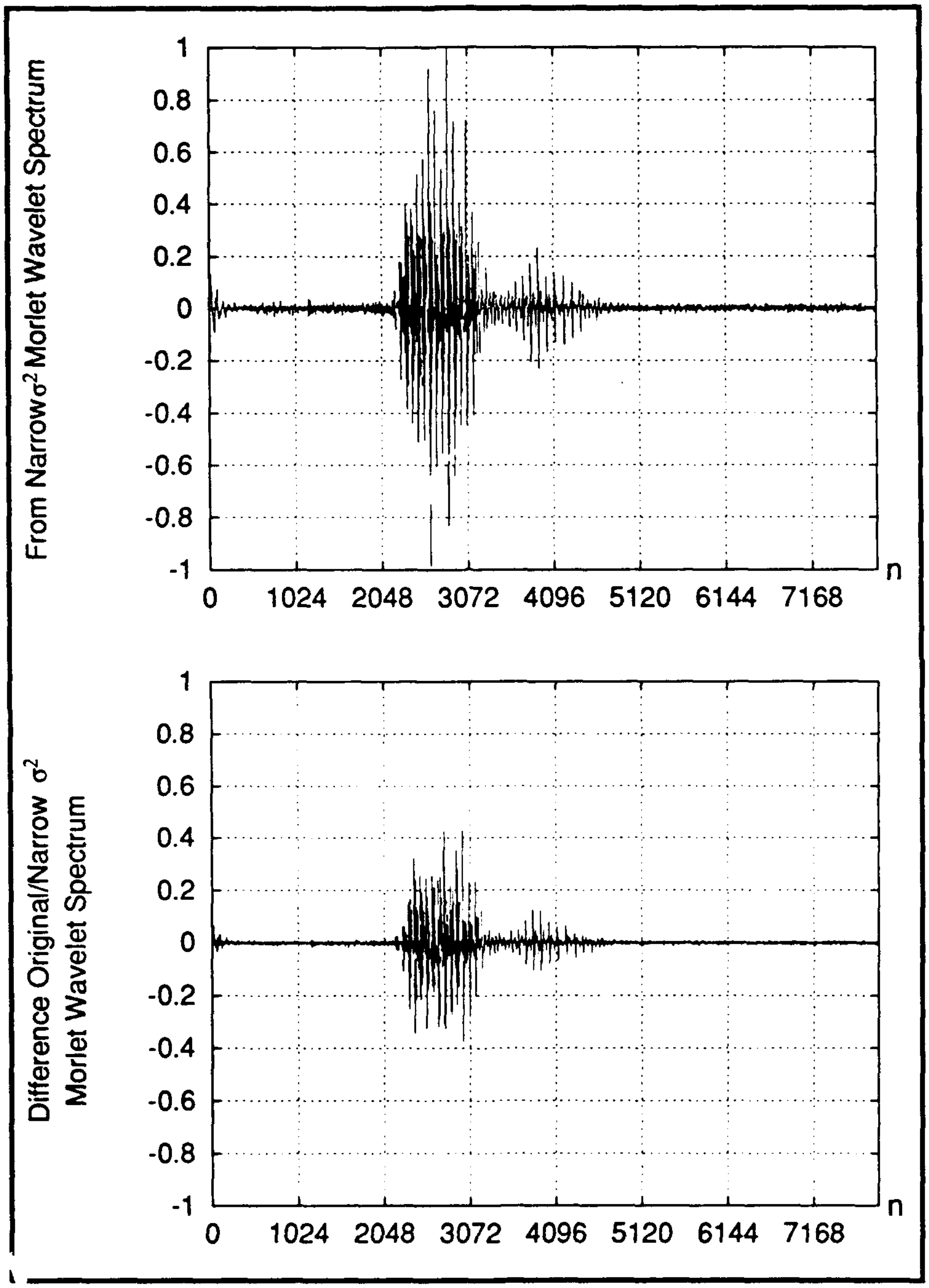

Figure 50. Reconstructed and Difference Signal from Narrow Variance Morlet Wavelet Coefficients

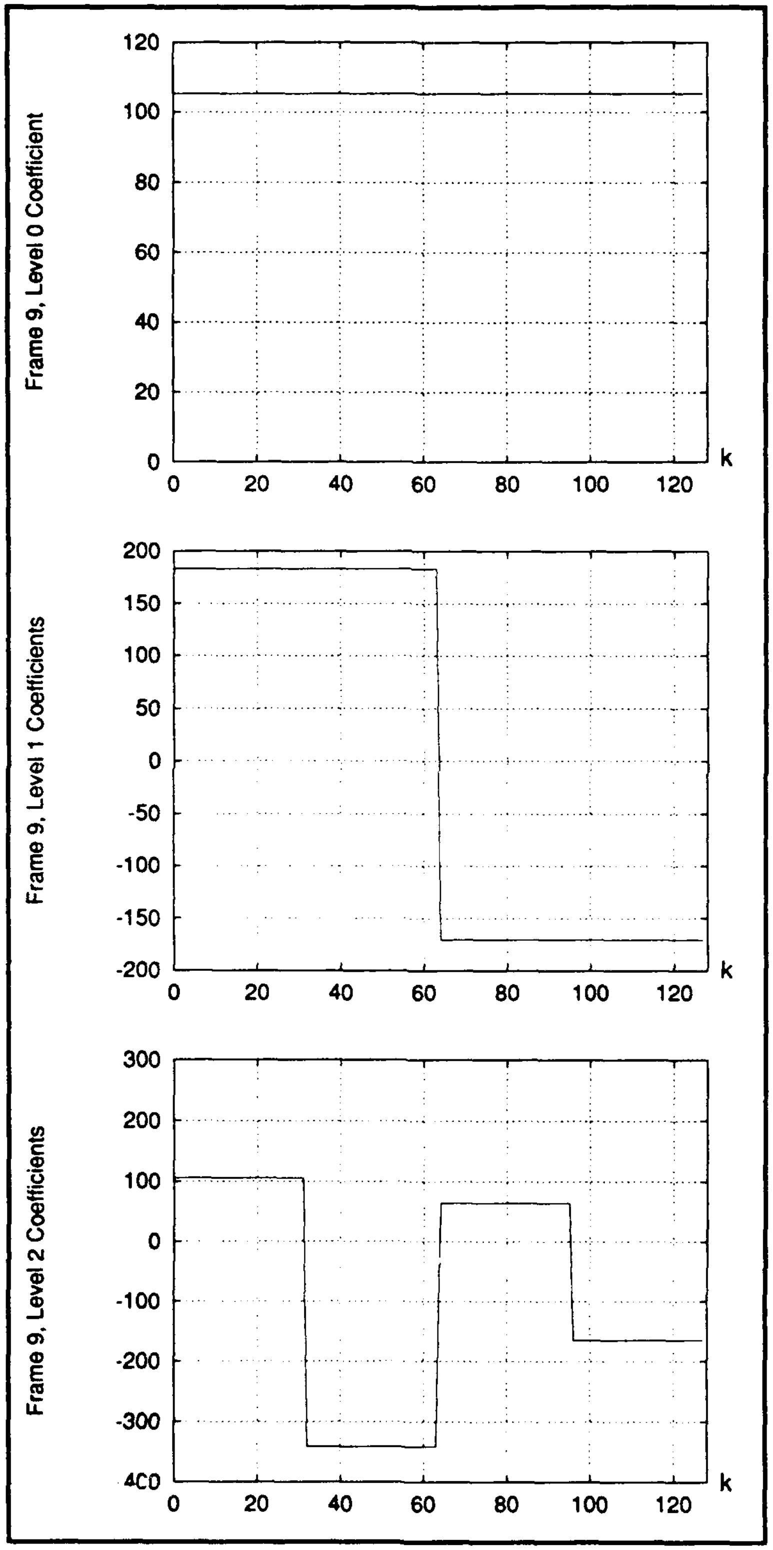

Figure 51. Levels 0–2 of Frame 9 of Haar Wavelet Decomposition

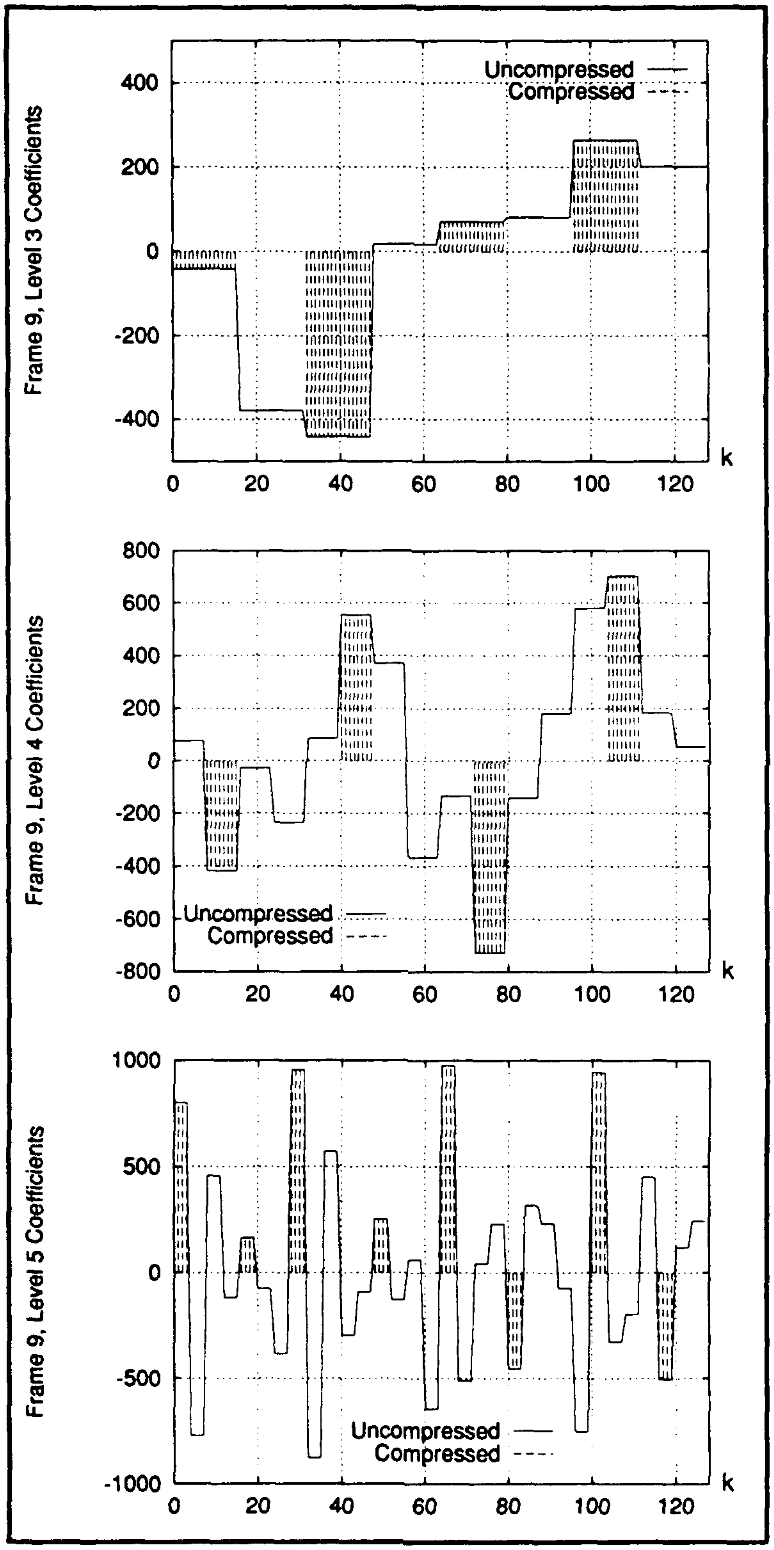

Figure 52. Levels 3–5 of Frame 9 of Haar Wavelet Decomposition

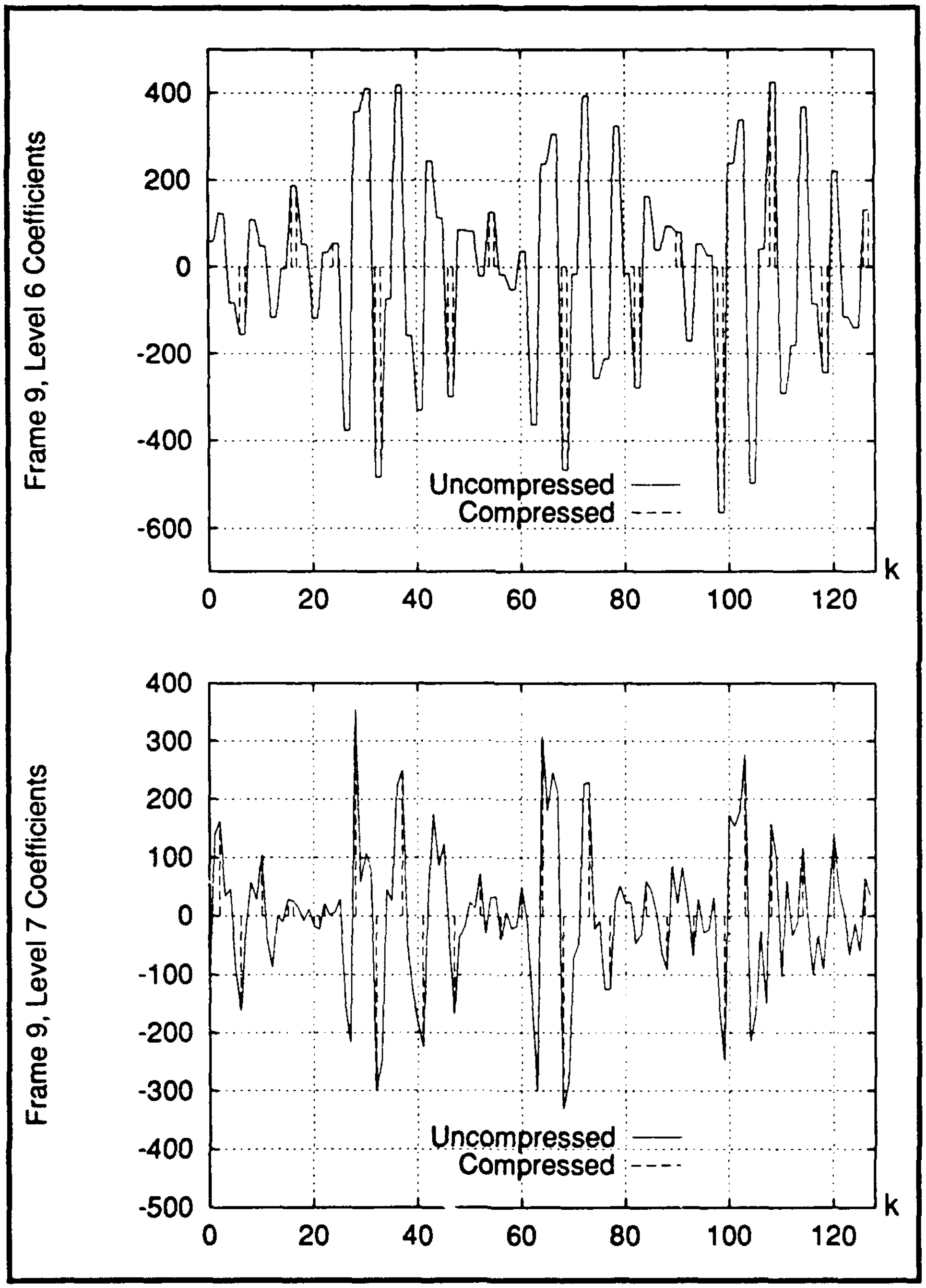

Figure 53. Levels 6–7 of Frame 9 of Haar Wavelet Decomposition

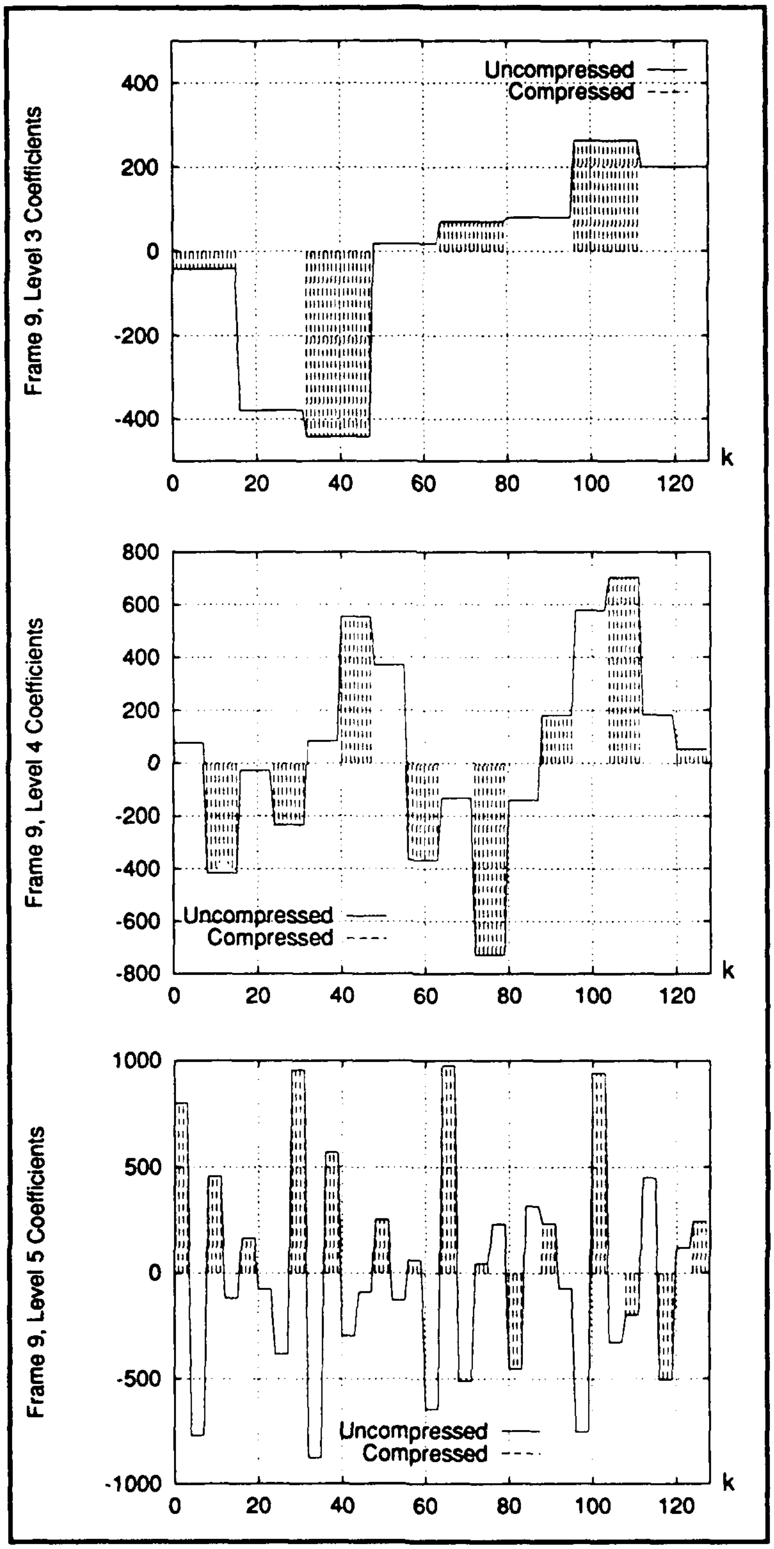

Figure 54. Levels 3–5 of Frame 9 Nearest Neighbor LIN Compression Result

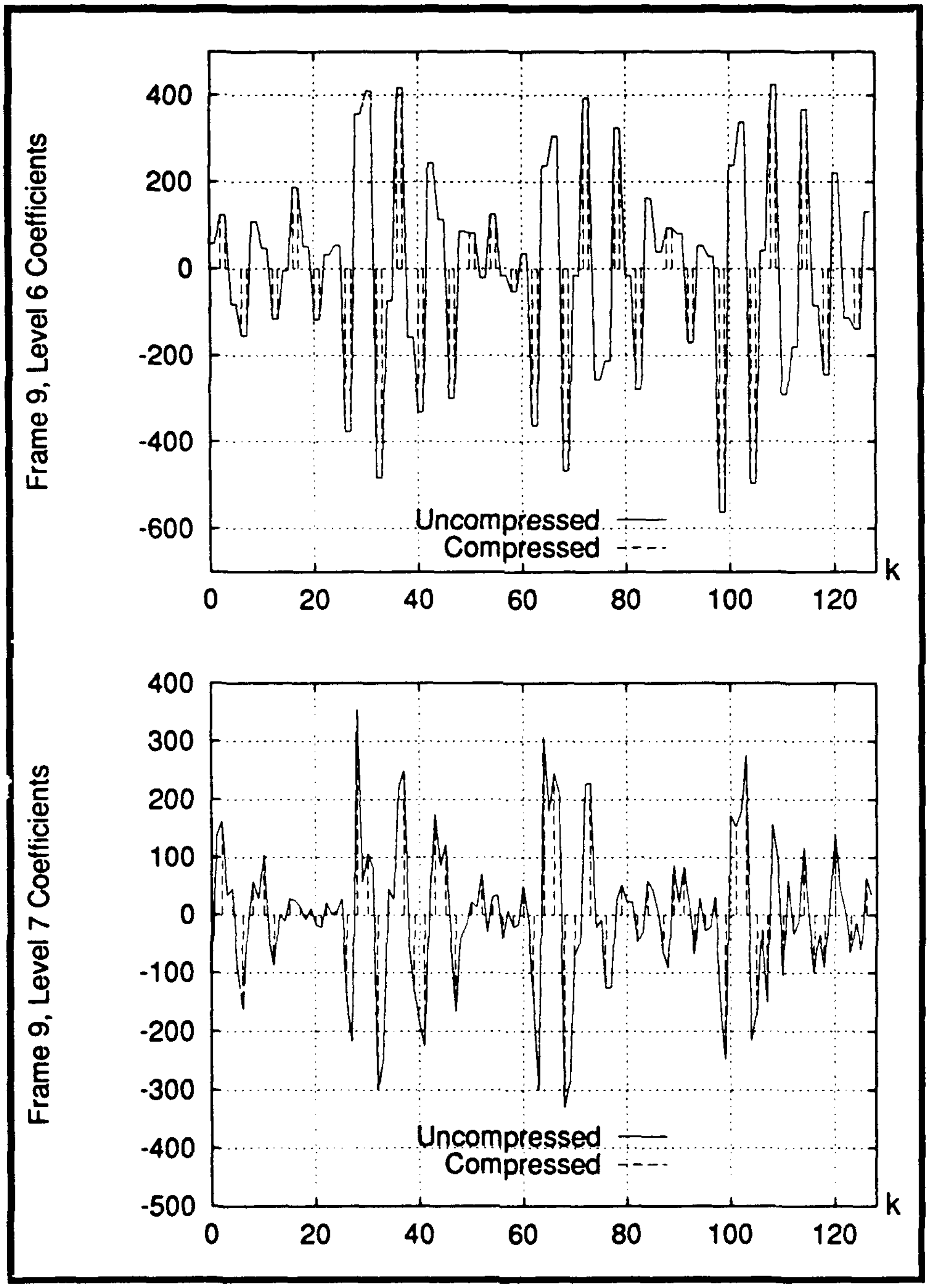

Figure 55. Levels 6–7 of Frame 9 Nearest Neighbor LIN Compression Result

Noise Filtering Analysis

Preliminary tests suggest that the use of LINs for noise filtering neither enhances nor degrades the intelligibility of the speech signal. With high signal-to-noise ratios (SNRs) the speech was equally intelligible before and after compression. Similarly, when the SNR was lowered significantly, the speech was not understood either before or after compression. Although this is not a dramatic result, it is significant when comparing with LPC coded speech. It is well known that LPC coded speech degrades very rapidly (nonlinearly) as a function of decreasing SNR. Thus, if the speech signal is understandable, no extra noise filtering is required before the speech is compressed with LINs as is required for LPC compressed speech.

Summary of Results

The results reported in this chapter were for the spoken word seven, and, again, these results are representative of the results found for the rest of the spoken digits.

In the case where speech is decomposed using the short-time Fourier and Gabor transforms, the LIN is capable of finding all relative maxima that are associated with the glottal frequency and its harmonics, and most importantly the formant peaks. When the Gabor decomposition was used, it was found that the quality of the reconstructed speech depends on the variable Q, which defines the extent of the biorthogonal function. Small values of Q are preferable than large values of Q in this case, but just the opposite is true if all the Gabor coefficients are used. A one-sided exponential window produces periodic clicks in the reconstructed signal, even when all coefficients are used. The reconstructed speech from the compressed short-time Fourier spectrum is of much better quality than the Gabor reconstructed speech. Of the three windows used in the short-time Fourier decomposition, the Gaussian window produced the best quality reconstructed speech. Speech reconstructed from the LIN compressed Gaussian spectrum using the ERB rule is virtually indistinguishable from the LIN compressed spectrum using the CB rule. The CB rule appears to define the limit of the competitive bandwidths since increasing the bandwidths beyond this limit significantly degrades the quality of the reconstructed speech. Overall, compression ratios between approximately 20 and 28 were achieved resulting in perfectly understandable speech. This amount of compression translates to well below 4.8 kbits/sec. speech.

The LIN, as designed in this thesis, does not compress the affine wavelet spectrum suitably for resynthesizing good quality speech. Even in efforts where compression was allowed in only one level, the resultant regenerated speech was sufficiently degraded in a manner that was consistently rated worse than the regenerated speech from the compressed short-time Fourier spectrum. In these cases the compression of the Fourier spectrum was nearly 20 times greater the the compression of the affine spectrum.

In the concluding chapter, the implications of these results, especially as they relate to the physiology of hearing, are explored. In addition, suggestions as to how the quality of the reconstructed speech can be improved from the compressed spectra explored in this thesis are offered.

V. Conclusions and Recommendations

Introduction

The main purpose of this research was to develop non-linear lateral inhibition networks (LINs) for coding and compressing the affine and W-H decompositions of speech. The choice of LINs for compression is biologically motivated since these networks are ubiquitous in sensory preprocessing systems across species for the main purpose of spatio/temporal contrast enhancement. In the W-H case, the LIN is designed to search for spectral peaks and eliminate the rest of the frequency components in each time slice of the two-dimensional transform lattice. In the affine wavelet transform space, the LIN is designed to pick temporal coefficient peaks in each level of frequency analysis. The LIN was tuned to approximate the overlapping bandwidths of the theoretical filter model of the cochlea.

In the following section, the most important results of this research are summarized. From these results, conclusions and recommendations for possible future research are suggested. This chapter and this thesis concludes with the biological implications of the results found in this research.

Summary of Results, Conclusions, and Recommendations

The best overall decomposition/compression/resynthesis results were obtained from the short-time Fourier domain. In that case, up to approximately 28 times compression of the short-time Fourier spectrum was achieved resulting in clearly intelligible speech. This amounted to approximately 95% elimination of the spectrum. The resynthesized speech compressed at these levels translates to under 4.8 kbits/sec. speech. Although the compressed speech is perfectly understandable, this speech is not of toll quality due to the musical quality of the fricatives. The compression ratios achieved—for a specific tuning of the LIN–were a function of the window used in the Fourier expansion.

The compactly supported Gaussian window produced the best compression, followed by the Hamming window, and the rectangular window in that order. If one examines the spectral characteristics of each window [30], one might conclude that this is due to the side lobe characteristics of each window. All other things being equal, the more the side lobes are suppressed, the more the LIN compresses. This hypothesis can be tested in future research by using other windows with even better side lobe characteristics than the compactly supported Gaussian window, in particular the Dolph-Chebyshev and Kaiser-Bessel windows.

Compressing the Gabor spectrum produced much noisier resynthesized speech than what was obtained from the short-time Fourier domain. It is concluded here, that the poor compression/resynthesis obtained from the Gabor decomposed speech is not due to the LIN compression. In each case, the LIN found peaks that were associated with the important characteristics of speech: the glottal frequency and its harmonics as well as the formant peaks. Therefore, the problem in the case of the Gabor expansion is not in the information the LIN is extracting but the fact that components are being eliminated. The conjecture is that errors arise due to problems in approximating and truncating the biorthogonal function of the window in the Gabor expansion. Further elimination of spectral components may produce Gibbs like errors that cannot be eliminated with windowing. Because of these errors and the extra computational complexity of the Gabor expansion, this representation is not recommended as an appropriate representation for speech compression.

The LIN compressed affine wavelet spectrum does not produce good quality resynthesized speech. Regardless of whether the Haar or Morlet wavelet was used to decompose speech, the signals regenerated from the compressed representations are quite rough sounding. This was found to be the case even when the LIN was tuned for minimal competition, that is, only nearest neighbor competition. It appears that Mallat's method [45] of regenerating the coefficients via convex space projections is necessary to reproduce good quality reconstruction. This process requires that at least the local maxima are chosen. The LIN chooses local maxima that do not necessarily coincide with the local maxima defined by Mallat. Therefore, if the LIN is to be used for this purpose, the network must be redesigned.

Even if a LIN can be designed to choose the appropriate maxima and the projection method is used to regenerate the speech signals, there appears to be at least two fundamental problems with the affine wavelet representation for speech compression. The first is the limitation of compression that may be achieved through the process described above. As may be seen from Figures 51 through 55, the high rate of variation in the speech signal produces a high rate of variation in the affine spectrum. As a result of this, relatively few components can be classified as non-maxima and thereby eliminated. As a rough estimate, only 25% of the coefficients of the speech signals processed in this thesis fall in that category. This is far less than the number of coefficients that can be eliminated in the short-time Fourier space. In addition, a real-time convex space projection system would require a highly parallelized and expensive system.

Because of the expense of the convex space projection method, a simple cubic spline or Hermite polynomial *first cut interpolation* of the coefficients should be tested. This may be all that is required to dramatically improve the reconstructed speech. This method should also be tried in the compressed short-time Fourier spectrum, for it might eliminate the musical quality of the the fricatives of the reconstructed speech.

There are other implementation issues which must also be addressed. In this thesis, the LIN was modeled as a system of non-linear differential equations whose solutions are approximated by Euler's method. This is not the most efficient method of implementation, however, it established a baseline. As may be recalled, in Chapter III, a linear approximation to the LIN may be implemented as overlapped Mexican-hat function digital filters. In fact, this can be reformulated as a wavelet process. This approach would significantly reduce the computational load. This suggests that an affine wavelet decomposition could be used on the short-time Fourier spectrum as the compression algorithm. A number of implementations are, therefore, possible when viewed in this manner. In spite of this, it must be realized that linear approximations of non-linear LIN dynamics will not produce the same computational results. Whether these approximations will yield results that sound acceptable to listeners should be established. In addition to the digital implementations, analog VLSI and CMOS implementations of LINs are also available [52, 41]. These circuits should satisfy the most stringent real-time requirements.

Biological Implications

In the previous chapter it was reported that nearly 95% of the frequency components of the short-time Fourier spectrum can be eliminated without producing a significant assault to the hearing mechanisms, whereas eliminating even a small number of coefficients in the affine wavelet spectrum produces errors that the hearing mechanism does not tolerate. What are the differences in the types of errors that occur when compressing the respective spectra, and how do these affect the hearing mechanism? The answer to the first question is straightforward. However, one must hypothesize when answering the second question.

It was discussed in Chapter III that time aliasing errors occur when the Fourier spectrum is nonuniformly sampled below the Nyquist rate. However, these errors can be suppressed with windowing. On the other hand, compressing the affine wavelet spectrum produces random or high frequency noise [45, 3]. The author of an Aware, Inc. technical report writes:

> Fourier-based spectral techniques [in compression] tend to produce errors of the aliasing type since the frequency spectrum itself is distorted, while wavelet methods tend to produce errors of the noisy type. Noise is far less offensive to the human visual system than aliasing. [3:40]

Evidently, just the opposite is true for the auditory system. Why this should be the case is now explored.

Until quite recently, the role of the efferent or *descending* auditory system was little more than a mystery. There remains a general lack of consensus as to the function of this system; however, recent hypotheses seem to be converging towards the following oversimplified process [2, 8, 29, 39, 69]. As may be recalled from Chapter II, acoustic signals produce frequency dependent vibratory motion at specific locations along the basilar membrane. This vibratory motion is transduced into electric potentials by the inner hair cells located on the basilar membrane. The electric potentials in turn activate the cochlear nerve fibers that ultimately signal the primary auditory fields of the neocortex via several brainstem and thalamic nuclei. This system is known as the *afferent* or *ascending* system. Thus, the afferent system acts as the primary auditory receptor system by transducing and encoding acoustic phenomena, and relaying that information to the brain.

The brain seems to actively control this process via the efferent system. A descending communication system, separate from but parallel to the ascending system, terminates in the brain at four nuclei—two on each side of the brain stem—called the lateral and medial superior olivary nuclei. These nuclei reside just above and ventral to the cochlear nucleus, the first relay station of the afferent system. Neurons from the olivary nuclei ultimately communicate with the cochlea through the olivocochlear bundle. The axons of this transmission line bifurcate and either synapse with the dendrites of the afferent system (near the inner hair cells) or synapse directly with the outer hair cells. Functionally, axodendritic stimulation appears to inhibit inner hair cell signaling. Outer hair cell stimulation produces much more complicated and interesting results. Evidence suggests that outer hair cell stimulation produces structural changes to the cell resulting in a mechanical action on the tectorial membrane. This electro-mechanical action acts as an automatic gain control of the afferent system. Kim elucidates with the following quotation

> The function of the OHCs [outer hair cells] is to enhance actively the sensitivity, tuning, and dynamic range of the mechanical response of the entire organ of Corti . . . , conferring high sensitivity, sharp tuning, and a wide dynamic range to the IHC [inner hair cell] subsystem. [39]

and Wiederhold adds:

> The efferents could also serve in a control system, possibly driven from cortical origins of the descending auditory pathways, to filter out acoustic signals that are distracting, or to make more intelligible those stimuli of particular interest. [69]

This last quotation may finally shed light on the question left unanswered in the beginning of this section: why are the errors produced by compressing the Fourier spectrum of speech tolerated by the hearing mechanism and not the errors produced by the compressed affine wavelet spectrum of speech? To aid the argument, Figure 56 shows, from top to bottom, the plots of the short-time Fourier spectrum of frame 18 (see previous chapter) of the original signal, the reconstructed signal from the compressed short-time Fourier spectrum using a Gaussian window, and the reconstructed signal from the compressed Morlet affine wavelet spectrum respectively. The middle plot shows that the process of reconstruction filled in some of the components eliminated in the compression process. Aliasing, then, amounts to localized energy reductions of the spectrum. On the other hand, the bottom plot verifies that broad band noise is produced when the affine wavelet spectrum is compressed. Thus, the conjecture is that the brain actively enhances the quality of the compressed Fourier spectrum reconstructed speech by boosting the gain at the appropriate locations along the basilar membrane. The brain is incapable of enhancing the compressed affine spectrum reconstructed speech to the same degree because the error is spread along the entire length of the basilar membrane. Therefore, the brain is unable to correctly enhance or attenuate the vibratory motion of the basilar membrane due to the improper cues. Psycho/physiological experiments can be designed and performed that test these hypotheses.

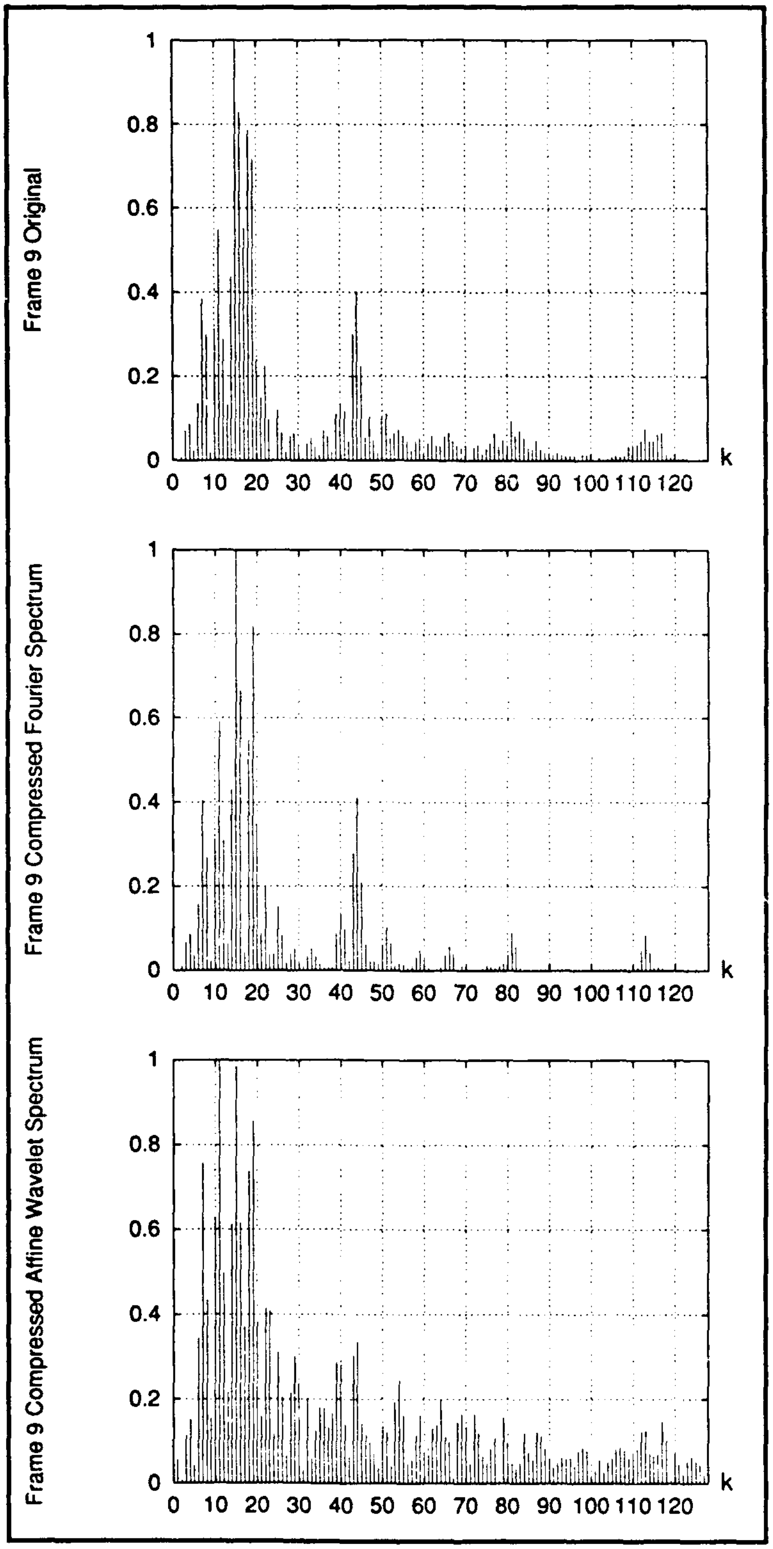

Figure 56. Respective Spectra of Original and Reconstructed Signals

Appendix A. *Code for Gabor Transform*

```
/***********************          gabor.m        ******************************/
/*                      Written by Rick Ricart, Capt, USAF                    */
/* Computes the Gabor representation of speech files using a Gaussian window */
/* The sound is sampled by the Codec at 8012.8 Hz. Each samle is, therefore, */
/* taken every 124.8E-06 sec. Delta t is chosen to be 256*124.8E-06 or       */
/* 31.949E-03 sec.  Delta f is, therefore, 31.3 Hz.                          */
/* This version uses 50% overlapped windows */
/* Call program as follows: gabor <filename> val, where val is 0 or 1        */
/* val=0 truncates biorthogonal function, val=1 leaves biorthogonal function */
/* at its maximum extent.                                                    */
/*****************************************************************************/

#import <sound/sound.h>
#import <math.h>
#import <dsp/arrayproc.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import <objc/objc.h>
#import <soundkit/Sound.h>
#import <soundkit/soundkit.h>
#import <string.h>
#import <macros.h>

#define N_DELTA_T 256
#define POINTS 2*N_DELTA_T
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 31
#define TOT M_TOTAL_DTS*N_DELTA_T  /* approximately one sec of sound */
#define TAU 1 /* exponential time constant */
#define PI 3.141592654
#define ABS(x) sqrt(pow(x,2.0))
#define ABS2(x) ((float)sqrt(((double)x)*((double)x)))


/* Global Variables */
id mySound, newSound;
SNDSoundStruct *soundStruct, *convertStruct;
/* the filenames */
char magnitudefile[80], phasefile[80], signalfile[80];


void get_data(short **temp,char *infile)
{
int error, data_size,i;
BOOL edit;
/*
    inPtr  = (short *) ((char *) inputSound   + inputSound->dataLocation);
    outPtr = (short *) ((char *) *outputSound + (*outputSound)->dataLocation);
*/
error = [mySound readSoundfile:infile];
/* initialize mySound to infile's mySound object */
soundStruct =[mySound soundStruct];
[mySound isEditable];
data_size = soundStruct->dataSize;

SNDAlloc(&convertStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
```

```
SNDConvertSound(soundStruct,&convertStruct);
*temp= (short *)(convertStruct+convertStruct->dataLocation);
}

void write_output_files(short *signal,float *gaborMag,
float *gaborPhase, float *gamma,
char *infile, char *index)
{
FILE *f1,*f2,*f3,*f4;
int k,m,n;
size_t s_len;

/* get length of input file not counting null terminator */
s_len=strlen(infile);
/* start name of signalfile with infile minus suffix .snd */
strncpy(signalfile,infile,(s_len-4));
/* must add terminating null, strncpy does not automatically do it */
signalfile[s_len-4]='\0';
/* start name of magnitudefile with infile minus suffex .snd */
strncpy(magnitudefile,infile,(s_len-4));
magnitudefile[s_len-4]='\0';
/* start name of phasefile with infile minus suffex .snd */
strncpy(phasefile,infile,(s_len-4));
phasefile[s_len-4]='\0';
/* add the appropriate gamma truncation index */
strcat(phasefile,index);
strcat(magnitudefile,index);
/* signalfile=filename.dat */
strcat(signalfile,".dat");
/* magnitudefile=filename_gaborMag.dat */
strcat(magnitudefile,"_gaborMag.dat");
/* phasefile=filename_gaborPhase.dat */
strcat(phasefile,"_gaborPhase.dat");

/* open and write signal data to signalfile */
if ((f1 = fopen(signalfile,"w")) == NULL){
       printf("\n*** Cannot can't create %s ***",signalfile);
       exit(0);
   }
loopn(TOT)
fprintf(f1,"%d\n",signal[n]);
/* open and write gaborMagnitude data to magnitudefile */
if ((f2 = fopen(magnitudefile,"w")) == NULL){
       printf("\n*** Cannot can't create %s ***",magnitudefile);
       exit(0);
   }
loopm(TOT)
fprintf(f2,"%f\n",gaborMag[m]);
/* open and write gaborPhase data to phasefile */
if ((f3 = fopen(phasefile,"w")) == NULL){
       printf("\n*** Cannct can't create %s ***",phasefile);
       exit(0);
   }
loopm(TOT)
fprintf(f3,"%f\n",gaborPhase[m]);

/* if ((f4 = fopen("gamma0.dat","w")) == NULL){
       printf("\n*** Cannot can't create %s ***","gamma0.dat");
       exit(0);
   }
loopm(2*TOT)
fprintf(f4,"%f\n",gamma[m]);*/
/* close all files */
fclose(f1);
fclose(f2);
fclose(f3);
/* fclose(f4);*/
}

/************** get_gamma() **************/
/* This function creates the biorthogonal function
of the Gaussian window */
/*****************************************/
void get_gamma(float *gamma, int index)
{
int i,n;
double k,coeff,time;
char in_string[8];


coeff=sqrt(sqrt(0.5));
k=1.85407468;
/* create biorthogonal window  function */
loopn(2*TOT){
time=((double)(n-TOT)/(double)(N_DELTA_T));
if((!index)&&((n<TOT-2*N_DELTA_T)||(n>=TOT+2*N_DELTA_T))){
if(n==0)
printf("truncated gamma function, index=0\n");
gamma[n]=0;
}
else{
if(n==0)
printf("max gamma function, index=1\n");
loopi(3)
gamma[n]+=(float)(coeff*
exp(PI*time*time-PI*
pow(floor(ABS(time)+0.5)+0.5+(double)i,2.0))*
pow((k/PI),(-3.0/2.0))*
pow(-1.0,floor(ABS(time)+0.5+(double)i)));
}
}

}

/************  window_data  **********************/
/* This function produces:
SUM(over q=0...M_DELTA_DTS) f(n+q*N_DELTA_T)*gamma(n+(q-m)*N_DELTA_T),
for a given time slice m, and where f is the signal function (timeReal) and
gamma is the biorthogonal function of the window */
/***********************************************/
void window_data(float *gamma, float *signal, float *timeReal)
{
# define A_ADR DSPAPGetLowestAddress() /* address of signal data part */
# define B_ADR (A_ADR + N_DELTA_T) /* address of biorthogonal data part */
# define C_ADR (B_ADR + N_DELTA_T) /* address of accumulated sum */
# define D_ADR (C_ADR + N_DELTA_T) /* address of result */
# define INC 1 /* increment for all array */

int p;

/* DSPFix24 arrays */
DSPFix24 A_ARRAY[N_DELTA_T],B_ARRAY[N_DELTA_T],RESULT[N_DELTA_T];

/* set vectors C and D to zero */
DSPAPvclear(C_ADR,INC,N_DELTA_T);
DSPAPvclear(D_ADR,INC,N_DELTA_T);

loopp(M_TOTAL_DTS){
/* Convert data from float to DSPFix24 */
DSPFloatToFix24Array(&signal[p*N_DELTA_T], A_ARRAY, N_DELTA_T);
DSPFloatToFix24Array(&gamma[p*N_DELTA_T], B_ARRAY, N_DELTA_T);

/* load arrays to DSP memory */
DSPAPWriteFix24Array(A_ARRAY, A_ADR, INC, N_DELTA_T);
DSPAPWriteFix24Array(B_ARRAY, B_ADR, INC, N_DELTA_T);

/* swap C and D arrays (answer to summing array) */
DSPAPvswap(C_ADR,INC,D_ADR,INC,N_DELTA_T);

/* This function computes A*B+C and puts answer in D */
DSPAPvtvpv(A_ADR,INC,B_ADR,INC,C_ADR,INC,D_ADR,INC,N_DELTA_T);
}

/* Return result from DSP memory to host memory */
DSPAPReadFix24Array(RESULT, D_ADR, INC, N_DELTA_T);

/* Convert data from DSPFix24 to float */
DSPFix24ToFloatArray(RESULT, timeReal, N_DELTA_T);

}



/* squash data */
float squash(float val, float oldmax, float oldmin, float newmax, float newmin)
{
float answer;

answer=(((val-oldmin)/(oldmax-oldmin))*(newmax-newmin))+newmin;
return answer;
}


void fft_wind_data(float *timeReal, float *timeImag, float *gaborReal,
float *gaborImag)
{
# define DATA_ADR DSPAPGetLowestAddressXY()
# define COEF_ADR (DATA_ADR + N_DELTA_T)
# define IMAG_DATA DSPMapPMemY(DATA_ADR)
# define REAL_DATA DSPMapPMemX(DATA_ADR)
# define SIN_TABLE DSPMapPMemY(COEF_ADR)
# define COS_TABLE DSPMapPMemX(COEF_ADR)

float *sinTab = DSPAPSinTable(N_DELTA_T);
float *cosTab = DSPAPCosTable(N_DELTA_T);
int i,m;

/* DSPFix24 arrays */
DSPFix24 TimeReal[N_DELTA_T],TimeImag[N_DELTA_T],GaborReal[N_DELTA_T],
GaborImag[N_DELTA_T];


/* Squash float data from -1 to 1 */
/* loopi(N_DELTA_T){
timeReal[i]=squash(timeReal[i],1E6,-1E6,1.0,-1.0);
timeImag[i]=squash(timeImag[i],1E6,-1E6,1.0,-1.0);
}*/

/* Convert data from float to DSPFix24 */
DSPFloatToFix24Array(timeReal, TimeReal, N_DELTA_T);
DSPFloatToFix24Array(timeImag, TimeImag, N_DELTA_T);

/* put the time domain complex array */
DSPAPWriteFix24Array(TimeReal, REAL_DATA, 1, N_DELTA_T);
DSPAPWriteFix24Array(TimeImag, IMAG_DATA, 1, N_DELTA_T);
```

```
/* put the cos and sine tables */
DSPAPWriteFloatArray(cosTab, COS_TABLE, 1, N_DELTA_T/2);
DSPAPWriteFloatArray(sinTab, SIN_TABLE, 1, N_DELTA_T/2);

DSPAPfftr2a(N_DELTA_T, DATA_ADR, COEF_ADR);

/*
 * Get the gabor domain complex array.
 * Tell monitor to read the array back with bit-reversed
 * addressing because fftr2a leaves its output shuffled.
 * Note that the skip factor (N_DELTA_T/2) is used to set the
 * DSP 'N' register.
 */
DSPSetDMAReadMReg(0);
DSPAPReadFix24Array(GaborReal, REAL_DATA, N_DELTA_T/2, N_DELTA_T);
DSPAPReadFix24Array(GaborImag, IMAG_DATA, N_DELTA_T/2, N_DELTA_T);
DSPSetDMAReadMReg(-1); /* re-select linear addressing */

/* Convert data from DSPFix24 to float */
DSPFix24ToFloatArray(GaborReal, gaborReal, N_DELTA_T);
DSPFix24ToFloatArray(GaborImag, gaborImag, N_DELTA_T);

/* Squash float data from -1 to 1 */
loopi(N_DELTA_T){
gaborReal[i]=squash(gaborReal[i],1.0,-1.0,1E6,-1E6);
gaborImag[i]=squash(gaborImag[i],1.0,-1.0,1E6,-1E6);
}


}


void get_gabor_mag(float *gaborReal, float *gaborImag, float *gaborMag)
{
int k,m;

loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
gaborMag[m*K_TOTAL_DFS+k]=
(float)sqrt((double)gaborReal[m*N_DELTA_T+k]*
(double)gaborReal[m*N_DELTA_T+k]+
(double)gaborImag[m*N_DELTA_T+k]*
(double)gaborImag[m*N_DELTA_T+k]);

}

void get_gabor_phase(float *gaborReal, float *gaborImag, float *gaborPhase)
{
int k,m;

loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
if((gaborReal[m*N_DELTA_T+k]==0)&&(gaborImag[m*N_DELTA_T+k]==0))
gaborPhase[m*K_TOTAL_DFS+k]=((float)PI/2);
else
gaborPhase[m*K_TOTAL_DFS+k]=
((float)atan2((double)gaborImag[m*N_DELTA_T+k],
(double)gaborReal[m*N_DELTA_T+k]));
}

main (int argc, char **argv)
{
short *temp; /* integer representation of original data */
float signal[TOT+N_DELTA_T/2]; /* float representation of original data */
float timeReal[2*TOT]; /* the overlapped windowed signal */
float timeImag[N_DELTA_T]; /* the imaginary portion of the data */
float gaborReal[2*TOT],gaborImag[2*TOT],gaborMag[TOT],gaborPhase[TOT];
float gamma[2*TOT]; /* the biorthogonal function of the window */
float temp_gamma[2*TOT]; /* squashed version of gamma */
int i;

/* initialize sound objects */
mySound=[Sound new];
newSound=[Sound new];
/* get sound data; timeReal has float rep. of sound */
get_data(&temp,argv[1]);
/* get float values of signal */
loopi(TOT)
signal[i]=(float)temp[i];
/* clear rest of signal */
for(i=TOT;i<TOT+N_DELTA_T/2;++i)
signal[i]=0;
/* zero out timeImag[] */
loopi(N_DELTA_T)
timeImag[i]=0;
/* clear gamma[] */
loopi(2*TOT)
gamma[i]=0;
/* generate biorthogonal window */
get_gamma(gamma,atoi(argv[2]));
/* Copy elements of data and gamma function to temporary arrays and
   squash float data from -1 to 1 */
loopi(TOT)
signal[i]=squash(signal[i],1E6,-1E6,1.0,-1.0);
loopi(2*TOT)
temp_gamma[i]=squash(gamma[i],200,-200,1.0,-1.0);
/* Initialize the DSP chip */
DSPAPInit();

/*  This computes Eq. (14) in the Einziger reference using the DSP chip
but with 50% overlap windows */
loopi(M_TOTAL_DTS*2){
if(i%2==0)
window_data(&temp_gamma[TOT-i*N_DELTA_T/2],signal,
&timeReal[i*N_DELTA_T]);
else
window_data(&temp_gamma[TOT-(i-1)*N_DELTA_T/2],&signal[N_DELTA_T/2],
&timeReal[i*N_DELTA_T]);
printf("windowing pass no. %d--%d\n",i,i%2);

}
/* This computes the FFT of each windowed time slice */
loopi(M_TOTAL_DTS*2){
fft_wind_data(&timeReal[i*N_DELTA_T],
timeImag,
&gaborReal[i*N_DELTA_T],
&gaborImag[i*N_DELTA_T]);
}

/* Free the DSP chip */
DSPAPFree();

get_gabor_mag(gaborReal,gaborImag,gaborMag);
get_gabor_phase(gaborReal,gaborImag,gaborPhase);
write_output_files(temp,gaborMag,gaborPhase,gamma,argv[1],argv[2]);

}
```

Appendix B. *Code for Short-Time Fourier Transform*

```
/************************        spectrogram.m        ************************/
/*                   Written by Rick Ricart, Capt, USAF                       */
/* Computes the spectrogram of arbitrary signals                              */
/* Call program as:  spectrogram filename.snd val1 val2                       */
/* Where val1=0 for a Rectangular window, val1=1 for a                        */
/* Gaussian window and val1=2 for a Hamming widow                             */
/* val2 is the standard deviation of the Gaussian noise                       */
/* If val2=0 no noise is added                                                */
/******************************************************************************/

#import <sound/sound.h>
#import <math.h>
#import <dsp/arrayproc.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import<objc/objc.h>
#import<soundkit/Sound.h>
#import<soundkit/soundkit.h>
#import<string.h>
#import<macros.h>



#define N_DELTA_T 256
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 66 /* Change this accordingly */
#define TOT  N_DELTA_T*M_TOTAL_DTS
#define PI 3.141592654
#define ABS(x)  ((float)sqrt(((double)x)*((double)x)))

/* Global Variables */
id mySound, newSound;
SNDSoundStruct *soundStruct, *convertStruct;
/* the filenames */
char spectrofile[80], phasefile[80], signalfile[80], newsignalfile[80],
newsoundfile[80];

/* Procedures  and Functions */
float gasdev();
void get_data(short **temp,char *infile)
{
int error, data_size,i;
BOOL edit;
/*
    inPtr  = (short *) ((char *) inputSound   + inputSound->dataLocation);
    outPtr = (short *)  ((char *) *outputSound + (*outputSound)->dataLocation);
*/
error = [mySound readSoundfile:infile];
/* initialize mySound to infile's mySound object */
soundStruct =[mySound soundStruct];
[mySound isEditable];
data_size = soundStruct->dataSize;
printf("data size is %d\n",data_size);
SNDAlloc(&convertStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
SNDConvertSound(soundStruct,&convertStruct);
*temp= (short *)(convertStruct+convertStruct->dataLocation);
```

```
printf("Finished get_data()\n");
}


void write_output_files(short *signal,float *newsignal,
float *Phase, float *spectrogram,
float *window, char *infile, char *wtype)
{
FILE *f1,*f2,*f3,*f4,*f5;
int k,m,n;
size_t s_len;

/* get length of input file not counting null terminator */
s_len=strlen(infile);
/* start name of newsignalfile with "new_(wtype)" */
strcpy(newsignalfile,"new_");
strcat(newsignalfile,wtype);
/* start name of signalfile with infile minus suffix .snd */
strncpy(signalfile,infile,(s_len-4));
/* must add terminating null, strncpy does not automatically do it */
signalfile[s_len-4]='\0';
/* start name of phasefile with infile minus suffix .snd */
strncpy(phasefile,infile,(s_len-4));
phasefile[s_len-4]='\0';
/* start name of spectrofile with infile minus suffix .snd */
strncpy(spectrofile,infile,(s_len-4));
spectrofile[s_len-4]='\0';
/* signalfile=filename.dat */
strcat(signalfile,".dat");
/* newsignalfile=new_filename.dat */
strcat(newsignalfile,signalfile);
/* add in window index */
strcat(phasefile,wtype);
strcat(spectrofile,wtype);
/* phasefile=filename_Phase.dat */
strcat(phasefile,"_Phase.dat");
/* spectrofile=filename_spectrogram.dat */
strcat(spectrofile,"_spectrogram.dat");

/* open and write signal data to signalfile */
if ((f1 = fopen(signalfile,"w")) == NULL){
        printf("\n*** Cannot can't create %s ***",signalfile);
        exit(0);
    }
loopn(TOT)
fprintf(f1,"%d\n",signal[n]);
/* open and write newsignal data to newsignalfile */
if ((f2 = fopen(newsignalfile,"w")) == NULL){
        printf("\n*** Cannot can't create %s ***",newsignalfile);
        exit(0);
    }
loopn(TOT)
fprintf(f2,"%f\n",newsignal[n]);
/* open and write Phase data to phasefile */
if ((f3 = fopen(phasefile,"w")) == NULL){
        printf("\n*** Cannot can't create %s ***",phasefile);
        exit(0);
    }
loopm(TOT)
fprintf(f3,"%f\n",Phase[m]);
/* open and write spectrogram data to spectrofile */
if ((f4 = fopen(spectrofile,"w")) == NULL){
        printf("\n*** Cannot can't create %s ***",spectrofile);
        exit(0);
    }
```

```
loopm(TOT)
fprintf(f4,"%f\n",spectrogram[m]);
/* if ((f5 = fopen("gauss_window.dat","w")) == NULL){
       printf("\n*** Cannot can't create %s ***","gauss_window.dat");
       exit(0);
    }
loopm(N_DELTA_T)
fprintf(f5,"%f\n",window[m]);*/

/* close all files */
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
/*fclose(f5);*/
}

void get_window(float *hamWindow, int window)
{
int n;
double time;

/* create window */
if(window==0)
loopn(N_DELTA_T)
hamWindow[n]=1.0; /* Rectangular */
else
if(window==1)
loopn(N_DELTA_T){
time=(dcuble)(n-N_DELTA_T/2)/(double)(N_DELTA_T);
hamWindow[n]=(float)(exp(-PI*4.0*time*time)); /* Gaussian */
}
else
if(window==2)
loopn(N_DELTA_T)
hamWindow[n]= (float)(0.54-0.46*   /* hamming */
cos((double)PI*(double)(2*n)/(double)(N_DELTA_T-1)));
else
if(window==3)
loopn(N_DELTA_T)
hamWindow[n]=  /* one period sin */
(float)(sin((double)PI*(double)n/(double)N_DELTA_T));
}

void ham_data(float *signal, float *hamWindow, float *timeReal)
{
int n;

/* multiply every N_DELTA_T points of data by window */
loopn(N_DELTA_T)
timeReal[n]=signal[n]*hamWindow[n];
}

/* squash data */
float squash(float val, float oldmax, float oldmin, float newmax, float newmin)
{
float answer;

answer=(((val-oldmin)/(oldmax-oldmin))*(newmax-newmin))+newmin;
return answer;
}


void fft_wind_data(float *timeReal, float *timeImag, float *gaborReal,
float *gaborImag)
```

```
{
# define DATA_ADR DSPAPGetLowestAddressXY()
# define COEF_ADR (DATA_ADR + N_DELTA_T)
# define IMAG_DATA DSPMapPMemY(DATA_ADR)
# define REAL_DATA DSPMapPMemX(DATA_ADR)
# define SIN_TABLE DSPMapPMemY(COEF_ADR)
# define COS_TABLE DSPMapPMemX(COEF_ADR)

float *sinTab = DSPAPSinTable(N_DELTA_T);
float *cosTab = DSPAPCosTable(N_DELTA_T);
int i,m;

/* DSPFix24 arrays */
DSPFix24 TimeReal[N_DELTA_T],TimeImag[N_DELTA_T],GaborReal[N_DELTA_T],
GaborImag[N_DELTA_T];


/* Squash float data from -1 to 1 */
loopi(N_DELTA_T){
timeReal[i]=squash(timeReal[i],1E6,-1E6,1.0,-1.0);
timeImag[i]=squash(timeImag[i],1E6,-1E6,1.0,-1.0);
}

/* Convert data from float to DSPFix24 */
DSPFloatToFix24Array(timeReal, TimeReal, N_DELTA_T);
DSPFloatToFix24Array(timeImag, TimeImag, N_DELTA_T);

/* put the time domain complex array */
DSPAPWriteFix24Array(TimeReal, REAL_DATA, 1, N_DELTA_T);
DSPAPWriteFix24Array(TimeImag, IMAG_DATA, 1, N_DELTA_T);

/* put the cos and sine tables */
DSPAPWriteFloatArray(cosTab, COS_TABLE, 1, N_DELTA_T/2);
DSPAPWriteFloatArray(sinTab, SIN_TABLE, 1, N_DELTA_T/2);

DSPAPfftr2a(N_DELTA_T, DATA_ADR, COEF_ADR);

/*
 * Get the gabor domain complex array.
 * Tell monitor to read the array back with bit-reversed
 * addressing because fftr2a leaves its output shuffled.
 * Note that the skip factor (N_DELTA_T/2) is used to set the
 * DSP 'N' register.
 */
DSPSetDMAReadMReg(0);
DSPAPReadFix24Array(GaborReal, REAL_DATA, N_DELTA_T/2, N_DELTA_T);
DSPAPReadFix24Array(GaborImag, IMAG_DATA, N_DELTA_T/2, N_DELTA_T);
DSPSetDMAReadMReg(-1); /* re-select linear addressing */

/* Convert data from DSPFix24 to float */
DSPFix24ToFloatArray(GaborReal, gaborReal, N_DELTA_T);
DSPFix24ToFloatArray(GaborImag, gaborImag, N_DELTA_T);

/* Squash float data from -1 to 1 */
loopi(N_DELTA_T){
gaborReal[i]=squash(gaborReal[i],1.0,-1.0,6E3,-6E3);
gaborImag[i]=squash(gaborImag[i],1.0,-1.0,6E3,-6E3);
}


}

void get_phase(float *spectroReal, float *spectroImag, float *Phase)
{
int k,m;
```

```
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
if((spectroReal[m*N_DELTA_T+k]==0)&&(spectroImag[m*N_DELTA_T+k]==0))
Phase[m*K_TOTAL_DFS+k]=((float)PI/2);
else
Phase[m*K_TOTAL_DFS+k]=
((float)atan2((double)spectroImag[m*N_DELTA_T+k],
(double)spectroReal[m*N_DELTA_T+k]));
}

void get_spectrogram(float *spectroReal,float *spectroImag, float *spectrogram)
{
int k,m;

loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
spectrogram[m*K_TOTAL_DFS+k]=
(float)sqrt((double)spectroReal[m*N_DELTA_T+k]*
(double)spectroReal[m*N_DELTA_T+k]+
(double)spectroImag[m*N_DELTA_T+k]*
(double)spectroImag[m*N_DELTA_T+k]);

}

void generate_tables(float *cosine,float *sine)
{
int k,m,n;
double temp;

/* create sine and cosine tables */
loopk(K_TOTAL_DFS)
loopn(N_DELTA_T){
cosine[k*N_DELTA_T+n]=(float)(cos(PI*(double)(2*k*n)
/(double)N_DELTA_T));
sine[k*N_DELTA_T+n]=(float)(sin(PI*(double)(2*k*n)
/(double)N_DELTA_T));
}
}

void regenerate_sound(float *Real, float *Imag,
 float *new_sound1, float *new_sound2,
 float *cosine, float *sine)
{
int i,k,m,n;

loopm(M_TOTAL_DTS){
/* printf("regenerating time slice %d ...\n",m); */
loopk(K_TOTAL_DFS)
loopi(N_DELTA_T)
new_sound1[m*N_DELTA_T+i]+=
(Real[2*m*N_DELTA_T+k]*
cosine[k*N_DELTA_T+i]-
Imag[2*m*N_DELTA_T+k]*
sine[k*N_DELTA_T+i]);
}
loopm(M_TOTAL_DTS){
/* printf("regenerating time slice %d ...\n",m); */
loopk(K_TOTAL_DFS)
loopi(N_DELTA_T)
new_sound2[m*N_DELTA_T+i]+=
(Real[((2*m)+1)*N_DELTA_T+k]*
cosine[k*N_DELTA_T+i]-
Imag[((2*m)+1)*N_DELTA_T+k]*
sine[k*N_DELTA_T+i]);
}
}

void ham_Newdata(float *new_sound1,float *new_sound2,
float *hamWindow,char *infile, int window)
{
int k,m,n;

/* multiply every N_DELTA_T points of data by window */
/* loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound1[m*N_DELTA_T+n]*=hamWindow[n];*/
/* mulitply every N_DELTA_T points  by window */
/* loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound2[m*N_DELTA_T+n]*=hamWindow[n]; */
/* if window not rect, add the overlapped reconstructed data points */
if(window) { /* if window is not rectangular... */
printf("Adding overlapped sample; not rectangular\n");
loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound1[m*N_DELTA_T+n+N_DELTA_T/2]+=
new_sound2[m*N_DELTA_T+n];
}
else
printf("Not adding overlapped samples; rectangular\n");
/* look for overflow data points */
   k=0;
loopn(TOT)
if(ABS(new_sound1[n])>32767.0)
k +=1;
printf("\nI found %d overflow data points processing %s\n",k,infile);
}

void add_noise(float *signal,float sdev)
{
int i,idum;
/* Noise power is the variance or sdev*sdev of Gaussian noise */
float temp;

/* set variance */
idum = -1;

loopi(TOT)
signal[i] +=sdev*gasdev(&idum);
}


main (int argc, char **argv)
{
short *temp; /* integer representation of original data */
float *timeReal; /* windowed overlapped signal data*/
float *signal;
float *timeImag; /* the imaginary portion of the data */
float *Phase;
float *spectroReal, *spectroImag, *spectrogram;
float *hamWindow;
float *new_sound1,*new_sound2;
float *cosine; /* the cosine table */
float *sine;  /* the sine table */
size_t s_len;
int i;

MALLOC(timeReal,2*TOT,float,"timeReal");
MALLOC(signal,TOT+N_DELTA_T/2,float,"signal");
MALLOC(timeImag,N_DELTA_T,float,"timeImag");
MALLOC(Phase,TOT,float,"Phase");
MALLOC(spectroReal,2*TOT,float,"spectroReal");
MALLOC(spectroImag,2*TOT,float,"spectroImag");
MALLOC(spectrogram,TOT,float,"spectrogram");
MALLOC(hamWindow,N_DELTA_T,float,"hamWindow");
MALLOC(new_sound1,TOT+N_DELTA_T/2,float,"new sound1");
MALLOC(new_sound2,TOT,float,"new sound1");
MALLOC(cosine,K_TOTAL_DFS*N_DELTA_T,float,"cosine table");
MALLOC(sine,K_TOTAL_DFS*N_DELTA_T,float,"sine table");
mySound=[Sound new];
newSound=[Sound new];
loopi(TOT){
new_sound1[i]=0;
new_sound2[i]=0;
}
/* get sound data; timeReal has float rep. of sound */
get_data(&temp,argv[1]);
/* copy and change integer sound samples to floats */
loopi(TOT)
signal[i]= (float)temp[i];
printf("Accessed data!\n");
/* add AWGN to signal */
if(atoi(argv[3])!=0)
add_noise(signal,atof(argv[3]));
/* pad rest of signal with zeroes */
for(i=TOT;i<TOT+N_DELTA_T/2;++i){
signal[i]=0;
new_sound1[i]=0;
}
/* compute hamming window */
get_window(hamWindow,atoi(argv[2]));
/*  This multiplies signal times window with 50% overlap */
/************ change ham_data to array processor routine on NeXT ****/
loopi(M_TOTAL_DTS*2){
ham_data(&signal[i*N_DELTA_T/2],hamWindow,
&timeReal[i*N_DELTA_T]);
printf("windowing pass no. %d\n",i);

}

loopi(N_DELTA_T)
timeImag[i]=0;
/* compute the dft of the windowed data */
DSPAPInit();
loopi(M_TOTAL_DTS*2){
fft_wind_data(&timeReal[i*N_DELTA_T],
timeImag,
&spectroReal[i*N_DELTA_T],
&spectroImag[i*N_DELTA_T]);
printf("    spectrogram pass no. %d\n",i);
}
DSPAPFree();
get_phase(spectroReal,spectroImag,Phase);
get_spectrogram(spectroReal,spectroImag,spectrogram);
generate_tables(cosine,sine);
regenerate_sound(spectroReal,spectroImag,new_sound1,new_sound2,
cosine,sine);
ham_Newdata(new_sound1,new_sound2,hamWindow,argv[1],atoi(argv[2]));
write_output_files(temp,new_sound1,Phase,spectrogram,
hamWindow,argv[1],argv[2]);
/* if there are any overflow data points truncate to + or- 32767 */
loopi(TOT)
temp[i]=(ABS(new_sound1[i])>32767)?
((short)(new_sound1[i]/ABS(new_sound1[i])*32767.0));
```

```
((short)new_sound1[i]);
/* get length of input file not counting null terminator */
s_len=strlen(argv[1]);
/* start name of newsoundfile with "new_" */
strcpy(newsoundfile,"new_");
strcat(newsoundfile,argv[2]);
/* concatenate input file name to "new_" in newsoundfile[] */
strcat(newsoundfile,argv[1]); /* newsound[]=new_filename.snd */
SNDWriteSoundfile(newsoundfile,convertStruct);



}
```

Appendix C. *Code for Reconstruction Algorithm from Short-Time Fourier Spectrum*

```
/************************        regenerate.m       *************************/
/*              Written by Rick Ricart, Capt, USAF                           */
/* This program reconstructs signals from the LIN output of gabor magnitudes */
/* This version is for overlapped coefficients                               */
/*****************************************************************************/

#import <sound/sound.h>
#import <dsp/arrayproc.h>
#import <math.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import<objc/objc.h>
#import<soundkit/Sound.h>
#import<soundkit/soundkit.h>
#import<string.h>
#import<macros.h>

#define N_DELTA_T 256
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 78
#define TOT N_DELTA_T*M_TOTAL_DTS  /* approximately one sec of sound */
#define TAU 1 /* exponential time constant */
#define PI 3.141592654
#define ABS(x) ((float)sqrt(((double)x)*((double)x)))


/* Global Variables */
id mySound;
SNDSoundStruct *soundStruct,*convertStruct;

char newsoundfile[80],newsignalfile[80];


/* Procedures  and Functions */
void get_soundstruct();
void get_data();
void find_energy();
void get_real_imag();
void write_output_files();
void generate_tables();
void regenerate_sound();
void get_window();
void ham_Newdata();
void output_newSpect();
/*These are needed for splines */
void spline();
void splint();
float *vector();

main (int argc, char **argv)
{
FILE *f1;
float *new_sound1,*new_sound2;
short *temp; /* integer representation of reconstructed signal */
```

```
float *spectroReal, *spectroImag, *spectrogram;
float *new_spectrogram,*Phase;
float *hamWindow;
float *cosine; /* the cosine table */
float *sine;  /* the sine table */
int i,size;
size_t s_len;

MALLOC(spectroReal,2*TOT,float,"spectroReal");
MALLOC(spectroImag,2*TOT,float,"spectroImag");
MALLOC(spectrogram,TOT,float,"spectrogram");
MALLOC(new_spectrogram,TOT,float,"new spectrogram");
MALLOC(Phase,TOT,float,"Phase");
MALLOC(hamWindow,N_DELTA_T,float,"hamWindow");
MALLOC(new_sound1,TOT+N_DELTA_T/2,float,"new sound1");
MALLOC(new_sound2,TOT,float,"new sound1");
MALLOC(cosine,K_TOTAL_DFS*N_DELTA_T,float,"cosine table");
MALLOC(sine,K_TOTAL_DFS*N_DELTA_T,float,"sine table");
loopi(TOT){
new_sound1[i]=0;
new_sound2[i]=0;
}
/* pad rest of signal with zeroes */
for(i=TOT;i<TOT+N_DELTA_T/2;++i)
new_sound1[i]=0;
get_data(spectrogram,new_spectrogram,Phase,argv[1],argv[2],argv[3]);
/* find_energy(new_spectrogram);*/
get_real_imag(spectrogram,new_spectrogram,Phase,spectroReal,spectroImag);
generate_tables(cosine,sine);
regenerate_sound(spectroReal,spectroImag,new_sound1,new_sound2,
cosine,sine);
/* compute hamming window */
get_window(hamWindow);
ham_Newdata(new_sound1,new_sound2,hamWindow,argv[1]);
mySound=[Sound new];
get_soundstruct(&temp);
size=soundStruct->dataSize;
/* clear old sound */
loopi(size)
temp[i]=0;
loopi(TOT)
temp[i]=(ABS(new_sound1[i])>32767)?
((short)(new_sound1[i]/
ABS(new_sound1[i])*32767.0)):
((short)new_sound1[i]);
/* loopi(TOT)
temp[i]=(short)(cos(PI*(double)(2*20*i)/(double)N_DELTA_T)*10000);*/
/* get length of input file not counting null terminator */
s_len=strlen(argv[3]);
/* start name of newsoundfile and newsignalfile with "short_" */
strcpy(newsoundfile,"short_");
strcpy(newsignalfile,"short_");
/* concatenate name of original sound file minus "_Phase.dat" identifier */
strncat(newsoundfile,argv[3],(s_len-10));
strncat(newsignalfile,argv[3],(s_len-10));
strcat(newsoundfile,".snd");
strcat(newsignalfile,".dat");
printf("The new sound file is %s\n",newsoundfile);
SNDWriteSoundfile(newsoundfile,convertStruct);
/* open and write signal data to newsignalfile */
if ((f1 = fopen(newsignalfile,"w")) == NULL){
       printf("\n*** Cannot can't create %s ***",newsignalfile);
       exit(0);
    }
loopi(TOT)
```

```
fprintf(f1,"%f\n",new_sound1[i]);
 fclose(f1);

}

void get_soundstruct(short **temp)
{
int error,data_size,i;
BOOL edit;
/*
    inPtr  = (short *) ((char *) inputSound   + inputSound->dataLocation);
    outPtr = (short *) ((char *) *outputSound + (*outputSound)->dataLocation);
*/
error = [mySound readSoundfile:"blank.snd"];
/* initialize mySound to infile's mySound object */
soundStruct =[mySound soundStruct];
data_size = soundStruct->dataSize;
SNDAlloc(&convertStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount," ");
SNDConvertSound(soundStruct,&convertStruct);
*temp= (short *)(convertStruct+convertStruct->dataLocation);
printf("Leaving get_soundstruct()\n");
}



void get_data(float *gaborMag,float *new_gaborMag,float *gaborPhase,
char *infile1,char *infile2,char *infile3)
{
int i;
FILE *f1,*f2,*f3;

printf("the three files are %s, %s, and %s\n",infile1,infile2,infile3);
if((f1 = fopen(infile1,"r"))==NULL){
printf("\n*** I can't read %s ***",infile1);
exit(1);
}
if((f2 = fopen(infile2,"r"))==NULL){
printf("\n*** I can't read %s ***",infile2);
exit(1);
}
if((f3 = fopen(infile3,"r"))==NULL){
printf("\n*** I can't read %s ***",infile3);
exit(1);
}
for(i=0;(fscanf(f1,"%f",&gaborMag[i])!=EOF);++i); /* reading gaborMag */
for(i=0;(fscanf(f2,"%f",&new_gaborMag[i])!=EOF);++i);
for(i=0;(fscanf(f3,"%f",&gaborPhase[i])!=EOF);++i);

fclose(f1);
fclose(f2);
fclose(f3);
/* printf("Leaving get_data\n");*/
}

void find_energy(float *new_gaborMag)
{
int k,m,n;
float max,energy,threshold;

/* clear variable max */
max=0.0;
/* set average threshold energy */
threshold=0.05;
/* determine the max value in current time slice */
```

```
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
max=(new_gaborMag[m*K_TOTAL_DFS+k]>max)?
(new_gaborMag[m*K_TOTAL_DFS+k]):max;
/* printf("The max value is %7.4g\n",max);*/
/* divide by max value of new_gaborMag[] */
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
new_gaborMag[m*K_TOTAL_DFS+k] /=max;
/* find the energy in each ~15msec. window */
loopm(M_TOTAL_DTS*4){
n=0;
energy=0.0;
loopk(K_TOTAL_DFS/2)
if(new_gaborMag[m*K_TOTAL_DFS/2+k]!=0){
++n;
energy += new_gaborMag[m*K_TOTAL_DFS/2+k];
}
energy /=(float)n;
/* printf("window %d--energy %7.4g--survivors %d\n",m,energy,n);*/
/* if the energy is less than the threshold zero entire window */
if(energy<threshold)
loopk(K_TOTAL_DFS/2)
new_gaborMag[m*K_TOTAL_DFS/2+k]=0;
}
/* multiply by max value of new_gaborMag[] */
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
new_gaborMag[m*K_TOTAL_DFS+k] *=max;
}

void get_real_imag(float *gaborMag,float *new_gaborMag, float *gaborPhase,
float *gaborReal, float *gaborImag)
{
int m,k,i;
float *temp,*org;
/* the following are for spline, splint calls */
float *x_set,*y_set,*y2,new_y;
float ans;


MALLOC(temp,TOT,float,"temp spect");
MALLOC(org,TOT,float,"temp spect");
/* Use new_gaborMag (the truncated gabor coefficients) as an index */
/* Do this by zeroing out gaborMag[] components which correspond to */
/* zero components of new_gaborMag[] */
memcpy(org,gaborMag,TOT*sizeof(float));
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS)
if((new_gaborMag[m*K_TOTAL_DFS+k]==0)||(k<0))
/* if(k>100)*/
gaborMag[m*K_TOTAL_DFS+k]=0;
memcpy(temp,gaborMag,TOT*sizeof(float));
/* Fill in freq and phase components with spline interpolation */
/* x_set=vector(1,128);
y_set=vector(1,128);
y2=vector(1,128);
loopm(M_TOTAL_DTS*2){
i=0;
loopk(K_TOTAL_DFS)
if(gaborMag[m*K_TOTAL_DFS+k]!=0||k==0){
++i;
x_set[i]=(float)k;
y_set[i]=gaborMag[m*K_TOTAL_DFS+k];
}
```

```
x_set[i+1]=127.0;
y_set[i+1]=0.0;
spline(x_set,y_set,i,0.0,0.0,y2);
loopk(K_TOTAL_DFS){
splint(x_set,y_set,y2,i,(float)k,&ans);
gaborMag[m*K_TOTAL_DFS+k]=ans;
}
if(m==19)
loopk(K_TOTAL_DFS){
printf("%5.3g  %5.3g  %5.3g\n",temp[m*K_TOTAL_DFS+k],
gaborMag[m*K_TOTAL_DFS+k],
org[m*K_TOTAL_DFS+k]);
}
}*/
/* Get real and imaginary values from magnitude and phase */
loopm(M_TOTAL_DTS*2)
loopk(K_TOTAL_DFS){  /* if Gaussian window *4 */
gaborReal[m*N_DELTA_T+k]=4.0*gaborMag[m*K_TOTAL_DFS+k]*
(float)cos((double)gaborPhase[m*K_TOTAL_DFS+k]);
gaborImag[m*N_DELTA_T+k]=4.0*gaborMag[m*K_TOTAL_DFS+k]*
(float)sin((double)gaborPhase[m*K_TOTAL_DFS+k]);

}
/* printf("Leaving get_real_imag()\n");*/
}

void generate_tables(float *cosine,float *sine)
{
int k,m,n;
double temp;

/* create sine and cosine tables */
loopk(K_TOTAL_DFS)
loopn(N_DELTA_T){
cosine[k*N_DELTA_T+n]=(float)(cos(PI*(double)(2*k*n)
/(double)N_DELTA_T));
sine[k*N_DELTA_T+n]=(float)(sin(PI*(double)(2*k*n)
/(double)N_DELTA_T));
}
/* printf("Leaving generate_tables()\n");*/
}

void regenerate_sound(float *Real, float *Imag,
 float *new_sound1, float *new_sound2,
 float *cosine, float *sine)
{
int i,k,m,n;

loopm(M_TOTAL_DTS){
/* printf("regenerating time slice %d ...\n",m); */
loopk(K_TOTAL_DFS)
loopi(N_DELTA_T)
new_sound1[m*N_DELTA_T+i]+=
(Real[2*m*N_DELTA_T+k]*
cosine[k*N_DELTA_T+i]-
Imag[2*m*N_DELTA_T+k]*
sine[k*N_DELTA_T+i]);
}
loopm(M_TOTAL_DTS){
/* printf("regenerating time slice %d ...\n",m); */
loopk(K_TOTAL_DFS)
loopi(N_DELTA_T)
new_sound2[m*N_DELTA_T+i]+=
(Real[((2*m)+1)*N_DELTA_T+k]*
cosine[k*N_DELTA_T+i]-
Imag[((2*m)+1)*N_DELTA_T+k]*
sine[k*N_DELTA_T+i]);
}
/* printf("Leaving generate_sound()\n");*/
}

void get_window(float *hamWindow)
{
int n;
double time;

/* create Hamming window */
loopn(N_DELTA_T)
hamWindow[n]=(float)(0.54-0.46*
cos((double)PI*(double)(2*n)/(double)(N_DELTA_T-1)));
/* printf("Leaving get_window()\n");*/
}

/* change this to a dsp routine */
void ham_Newdata(float *new_sound1,float *new_sound2,
float *hamWindow,char *infile)
{
int k,m,n;

/* multiply every N_DELTA_T points of data by window */
loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound1[m*N_DELTA_T+n]*=hamWindow[n];
/* mulitply every N_DELTA_T points  by window */
loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound2[m*N_DELTA_T+n]*=hamWindow[n];
/* add the overlapped reconstructed data points */
loopm(M_TOTAL_DTS)
loopn(N_DELTA_T)
new_sound1[m*N_DELTA_T+n+N_DELTA_T/2]+=
new_sound2[m*N_DELTA_T+n];
/* look for overflow data points */
   k=0;
loopn(TOT)
if(ABS(new_sound1[n])>32767.0)
k +=1;
printf("\nI found %d overflow data points processing %s\n",k,infile);
}
```

Appendix D. *Code for LIN Compression of W-H Spectra*

```
/************************* COMPETE.C ******************************/
/*               Written by Rick Ricart, Capt, USAF               */
/* This program computes LIN competition of W-H frequency spaces */
/* using the ERB and CB criteria.  Call program as follow:        */
/*           compete <filename> <0 or 1> <threshold>              */
/* where 0 signifies ERB and 1 CB.                                */
/* Threshold is any value, v such that  0<v< 1, (e.g., .009)      */
/* result filenames are automatically generated                   */
/*****************************************************************/

#include <math.h>
#include <stdio.h>
#include <macros.h>

#define F0 0.25
#define TAU 0.1
#define EPS 1.0
#define MAX 1.0
#define MIN 1.0
#define COMPGAIN 10.0
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 70
#define SURVIVORS 50

#define SQUARE(x) ((x)*(x))
#define SIGMOID(x) (SQUARE(x))/(SQUARE(x)+F0)

FILE outfile[80],result[80];

typedef struct{
float input;
float activity;
float output;
int *competitors;
int num_competitors;
}nodes;

float squash(float val, float oldmax, float oldmin, float newmax, float newmin)
{
float answer;

answer=(((val-oldmin)/(oldmax-oldmin))*(newmax-newmin))+newmin;
return answer;
}

void write_output(float *gaborMag,char *infile,int size,int bw, char *thresh)
{
FILE *fi,*fopen();
int k,m,n;

printf("Starting to output files\n");
/* start name of outfile with "ERB or CB" */
if(!bw){
strcpy(outfile,"ERB");
 /* outfile[3]='\0';*/
}
else{
strcpy(outfile,"CB");
/* outfile[2]='\0';*/
}
strcat(outfile,thresh);
/* newsignalfile=new_infile */
strcat(outfile,infile);
/* open and write signal data to signalfile */
if ((fl = fopen(outfile,"w")) == NULL){
       printf("\n*** Cannot can't create %s ***",outfile);
       exit(0);
    }
loopn(size)
fprintf(fl,"%f\n",gaborMag[n]);
fclose(fl);
}

void write_result(float *gaborMag, char *infile)
{
FILE *fl,*fopen();
int k,m,n;

/* start name of outfile with "new_" */
strcpy(result,"result_");
/* newsignalfile=new_infile */
strcat(result,infile);

if ((fl = fopen(result,"w")) == NULL){
       printf("\n*** Cannot can't create %s ***",result);
       exit(0);
    }
/* loopm(M_TOTAL_DTS) for nonovelapped coefficients loop(M_TOTAL_DTS*2)for
  overlapped coefficients */
loopm(M_TOTAL_DTS*2){
fprintf(fl,"TIME %d:   ",m);
loopk(K_TOTAL_DFS)
if(gaborMag[m*K_TOTAL_DFS+k]!=0)
fprintf(fl,"%d,%5.2g  ",k,gaborMag[m*K_TOTAL_DFS+k]);
fprintf(fl,"\n");
}
fclose(fl);
}

void compute_activity(nodes *n_list,nodes *node)
{
  float a,exc,inh;
int i;

a=node->activity;
inh=0;
exc=node->output+node->input;
loopi(node->num_competitors)
inh+=n_list[node->competitors[i]].output;
node->activity=a+TAU*EPS*(-a+(MAX-a)*exc-(MIN+a)*inh*COMPGAIN);
}

void compute_output(nodes *node, float thresh)
{
float a;

a=node->activity;
/* if(a<0)
node->output=0;
else
if(a>=0&&a<=1)
node->output=a;
else
```

```
node->output=1;*/
node->output=(a-thresh<0)?0:(SIGMOID(a-thresh));
}

void initialize_competitors(nodes *node,int index,int size, int bw)
{
int i,j,num,count;
char in_string[8];

/* change these appropriately thru freq vals MUST BE AN EVEN NUMBER */
/* Make this a SWITCH statement */
if(!bw){  /* this does ERB competition */
if(index == 0)
printf("Doing ERB competition\n");
 if(index<30)  /* for ERB <40, CB<30 */
 num=2; /* for ERB 2, CB 4 */
 else
 if(index<58)  /* for ERB 58, CB 47 */
 num=4; /* for ERB 4, CB 6 */
 else
 if(index<75) /* for ERB 75, CB 60 */
 num=6; /* for ERB 6, CB 8 */
 else
 if(index<92) /* for ERB 91, CB 72 */
 num=8; /* for ERB 8, CB 10 */
 else
 if(index<106) /* for ERB 106, CB 84 */
 num=10; /* for ERB 10, CB 12 */
 else
 if(index<121) /* for ERB 121, CB 94 */
 num=12; /* for ERB 12, CB 14 */
 else
 num=14; /* for ERB 14, CB 16 */
}
else{   /* this does CB competition */
if(index == 0)
printf("Doing CB competition...\n");
 if(index<30)  /* for ERB <40, CB<30 */
 num=4; /* for ERB 2, CB 4 */
 else
 if(index<47)  /* for ERB 58, CB 47 */
 num=6; /* for ERB 4, CB 6 */
 else
 if(index<60) /* for ERB 75, CB 60 */
 num=8; /* for ERB 6, CB 8 */
 else
 if(index<72) /* for ERB 91, CB 72 */
 num=10; /* for ERB 8, CB 10 */
 else
 if(index<84) /* for ERB 106, CB 84 */
 num=12; /* for ERB 10, CB 12 */
 else
 if(index<94) /* for ERB 121, CB 94 */
 num=14; /* for ERB 12, CB 14 */
 else
 num=16; /* for ERB 14, CB 16 */
}
node->num_competitors=num;
MALLOC(node->competitors,num,int,"allocating num_competitors");
/* competitors of index less than index of current node */
count=1;
loopi(num/2)
if(index-i-1>=0)
node->competitors[i]=index-i-1;
else{
```

```
if(index-1-i+count==index)
count +=1;
node->competitors[i]=index-1-i+count;
count +=2;
}
/* competitors of index greater than index of current node */
count=1;
loopi(num/2)
if(index+1+i<=size-1)
node->competitors[i+num/2]=index+1+i;
else{
if(index+1+i-count==index)
count +=1;
node->competitors[i+num/2]=index+1+i-count;
count +=2;
}
/****************** print statements ****************/
/* printf("the current node is %d and the competitors are: ",index);
loopi(num)
printf("%d ",node->competitors[i]);
printf("\n");
printf("Hit ENTER to continue");
gets(in_string);*/
/*******************                ****************/
}



main(int argc, char **argv)
{
float *array,*outarray,max,min,temp;
int i,j,k,l,size,time,count,index,tot;
size_t s_len;
nodes *n_list;
char in_string[8];
FILE *fin,*fout;

/* open input file */
if((fin = fopen(argv[1],"r"))==NULL){
printf("\n*** I can't read %s ***",argv[1]);
exit(1);
}
/**************************************************************************/
/* for nonoverlapped gabor coefficients size=K_TOTAL_DFS*M_TOTAL_DTS,
for overlapped gabor coefficients size=K_TOTAL_DFS*M_TOTAL_DTS*2 */
/****************************************************************************/
printf("\n starting %s\n",argv[1]);
printf("Bandwidth %s, and threshold %s\n", argv[2], argv[3]);
size=K_TOTAL_DFS*M_TOTAL_DTS*2;
/* allocate memory */
MALLOC(array,size,float,"allocating  array");
MALLOC(outarray,size,float,"allocating  outarray");
MALLOC(n_list,K_TOTAL_DFS,nodes,"allocating n_list");
tot = 0;
/* copy values of input file into array */
for(i=0;(fscanf(fin,"%f",&array[i])!=EOF);++i);
/* loopi(size)
array[i]=(array[i]!=0)?((float)log10((double)array[i])):0;*/
/* determines which nodes compete */
loopi(K_TOTAL_DFS)
initialize_competitors(&n_list[i],i,K_TOTAL_DFS,atoi(argv[2]));
/* initialize total number of time increments for solving shunting equations */
time=100;
/*******************************************************************************/
/* This is the outer loop for each time slice. Within this loop, the max
```

```
magnitude of the Gabor coefficients per time slice is determined and
all frequency components in that time slice are squashed between 1 and -1.
These values are then passed to the LIN and the activity and outputs
are determined for each node in the LIN.  The result is then saved in
outarray[] */
/*******************************************************************************/
loopk(M_TOTAL_DTS*2){
/* clear variable max */
max=0.0;
/* determine the max value in current time slice */
loopi(K_TOTAL_DFS)
if(array[k*K_TOTAL_DFS+i]>max)
max=array[k*K_TOTAL_DFS+i];
printf("\nMax element is %f at time %d---Starting squash...\n",max,k);
/* squash all frequency components of current time slice between 1 and -1 */
loopi(K_TOTAL_DFS)
array[k*K_TOTAL_DFS+i]=squash(array[k*K_TOTAL_DFS+i],max,0.0,1.0,0.0);
/* clear activity and output variables of all nodes */
loopi(K_TOTAL_DFS){
n_list[i].activity=0.0;
n_list[i].output=0.0;
}
/* initialize input to all nodes */
loopi(K_TOTAL_DFS)
n_list[i].input=array[k*K_TOTAL_DFS+i];
/* compute the activity and output of all nodes in current time slice */
loopj(time){
loopi(K_TOTAL_DFS)
compute_activity(n_list,&n_list[i]);
loopi(K_TOTAL_DFS)
compute_output(&n_list[i],atof(argv[3]));
}
/* copy the result to outarray[] */
loopi(K_TOTAL_DFS)
outarray[k*K_TOTAL_DFS+i]=n_list[i].output;
/***** print statements ***********/
/* loopi(K_TOTAL_DFS){
printf("Node: %d Array: %7.4f Input: %7.4f
Activity: %7.4f Output: %7.4f\n",
i,array[k*K_TOTAL_DFS+i],n_list[i].input,
n_list[i].activity,n_list[i].output);
printf("Hit ENTER to continue");
gets(in_string);
}*/
/**********************************/
count =0;
/* unsquash result */
loopi(K_TOTAL_DFS){
temp=squash(outarray[k*K_TOTAL_DFS+i],1.0,0.0,max,0);
if(temp<10.0)
outarray[k*K_TOTAL_DFS+i]=0;
else{
outarray[k*K_TOTAL_DFS+i]=temp;
count +=1;
}
}
printf("In time slice %d there were %d survivors\n",k,count);
if(count>SURVIVORS){
loopj((count -=SURVIVORS)){
min=outarray[k*K_TOTAL_DFS];
index=0;
printf(" Cutting %d survivors to %d\n",count,SURVIVORS);
loopi(K_TOTAL_DFS){
if(outarray[k*K_TOTAL_DFS+i]<min){
min=outarray[k*K_TOTAL_DFS+i];
index=i;
}
}
outarray[k*K_TOTAL_DFS+index]=0;
}
}
tot +=count;
printf("Total now is %d\n",tot);

}
/*************************** end of main loop ******************************/
/******* print statements *********/
/* printf("ARRAY:\n");
loopi()
printf("%3.2f ",array[i]);
printf("\n");
printf("OUTPUT:\n");
loopi(K_TOTAL_DFS)
printf("%3.2f ",n_list[i].output);
printf("\n");
printf("OUTARRAY:\n");
loopi(size)
printf("%3.2f ",outarray[i]);*/
/***********************************/
write_output(outarray,argv[1],size,atoi(argv[2]),argv[3]);
printf("The average no. of survivors per window is %5.3g, and RC=%5.3g\n",
(float)tot/(float)(2*M_TOTAL_DTS),
(float)(2*M_TOTAL_DTS*K_TOTAL_DFS)/(float)tot);
/* write_result(outarray,argv[1]); */
fclose(fin);

}
```

Appendix E. *Code for Haar Wavelet Decomposition, Compression, and Reconstruction*

```
/***********************          wavelet.m        ***********************/
/*              Written by Rick Ricart, Capt, USAF                        */
/* This program computes the Haar wavelet decomposition,compression,     */
/* and reconstruction of a time signal                                   */
/*************************************************************************/


#import <sound/sound.h>
#import <math.h>
#import <dsp/arrayproc.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import <objc/objc.h>
#import <soundkit/Sound.h>
#import <soundkit/soundkit.h>
#import <string.h>
#import <macros.h>

#define N_DELTA_T 256
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 31
#define TOT M_TOTAL_DTS*N_DELTA_T  /* approximately one sec of sound */
#define PI 3.141592654
#define LEVELS 8
#define MAX 1
#define MIN 1
#define COMPGAIN 10.0
#define F0 0.25
#define TAU 0.1
#define EPS 1.0

#define ABS(x) sqrt(pow(x,2.0))
#define ABS2(x) ((float)sqrt(((double)x)*((double)x)))
#define SQUARE(x) ((x)*(x))
#define SIGMOID(x) (SQUARE(x))/(SQUARE(x)+F0)


/* Global Variables */
id mySound, newSound;
SNDSoundStruct *soundStruct, *convertStruct, *shortStruct;

/* the filenames */
char newsoundfile[80],shortsoundfile[80];

void compute_trans();
float compute_iproduct();

typedef struct grid_element {
struct grid_element *r_ptr;
struct grid_element *l_ptr;
struct grid_element *rleg_ptr;
struct grid_element *lleg_ptr;
struct grid_element *up_ptr;
float iproduct_val;
float input;
float activity;
float output;
int type;
} grid_element;

typedef struct {
grid_element *head, *current;
} list;

typedef struct {
float *wavelet;
} wavelets;

/******************** get_data() ************************************/
/* This procedure opens and reads the signal values of a sound file */
/*******************************************************************/
void get_data(short **temp,short **temp2,char *infile)
{
int error, data_size,i;
BOOL edit;
/*
    inPtr  = (short *) ((char *) inputSound   + inputSound->dataLocation);
    outPtr = (short *) ((char *) *outputSound + (*outputSound)->dataLocation);
*/
error = [mySound readSoundfile:infile];
/* initialize mySound to infile's mySound object */
soundStruct =[mySound soundStruct];
[mySound isEditable];
data_size = soundStruct->dataSize;
SNDAlloc(&convertStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
SNDAlloc(&shortStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
SNDConvertSound(soundStruct,&convertStruct);
SNDConvertSound(soundStruct,&shortStruct);
*temp= (short *)(convertStruct+convertStruct->dataLocation);
*temp2= (short *)(shortStruct+shortStruct->dataLocation);
}

list *make_grid()
{
list *temp;

MALLOC(temp,1,list,"Allocating list");
temp->head=NULL;
temp->current=NULL;
/* printf("Made make_grid successfully\n");*/
return temp;
}

void allocate_node(list *l, grid_element *node)
{
if(l->head==NULL){
l->head=node;
l->current=node;
node->l_ptr=node->r_ptr=NULL;
}
else{
node->l_ptr=l->current;
l->current->r_ptr=node;
l->current=node;
}
node->rleg_ptr=node->lleg_ptr=NULL;
node->up_ptr=NULL;
```

```
}

void allocate_grid_triad(list *l, grid_element *element)
{
grid_element *temp1,*temp2;

/* printf("\nAllocating a triad\n");*/
/* allocate both legs */
MALLOC(temp1,1,grid_element,"allocating lleg");
MALLOC(temp2,1,grid_element,"allocating rleg");
temp1->type=0;
temp2->type=1;
element->lleg_ptr=temp1;
element->rleg_ptr=temp2;
temp1->r_ptr=temp2;
temp2->l_ptr=temp1;
temp1->up_ptr=temp2->up_ptr=element;
if(element->l_ptr==NULL||element->l_ptr->rleg_ptr==NULL)
temp1->l_ptr=NULL;
else {
temp1->l_ptr=element->l_ptr->rleg_ptr;
element->l_ptr->rleg_ptr->r_ptr=temp1;
}
temp2->r_ptr=NULL;
temp1->rleg_ptr=temp2->rleg_ptr=temp1->lleg_ptr=temp2->lleg_ptr=NULL;
l->current=element;
/* printf("allocated grid triad successfully\n");*/
}


void inner_loop(int level, list *l, grid_element *element, wavelets *wl,
float *signal,float *new_signal,
int m, int points)
{
int elements;

elements=(int)pow(2.0,(double)(LEVELS-level));
if(level==0)
return;
else
if(element->r_ptr==NULL){
points -=elements;
/* printf("Moving up a level, moving element ptr up, and inner loop.\n");*/
return inner_loop(--level,l,element=element->up_ptr,
wl,signal,new_signal,m,points);
}
else{
/* printf("Moving element ptr right and allocating another triad\n");*/
points +=elements;
compute_trans(level,l,element=element->r_ptr,wl,signal,
new_signal,m,points);
}
}

void regenerate_signal(float *new_signal, wavelets *wl,int elements,
int points, int m, float mag)
{
int i;

loopi(elements){
new_signal[m*N_DELTA_T+i+points] +=wl->wavelet[i]*mag;
/* printf("%5.3g\n",new_signal[m*N_DELTA_T+i+points]);*/
}
}
```

```
/******************** compute_trans() ****************************/

/*****************************************************************/

void compute_trans(int level, list *l, grid_element *element, wavelets *wl,
float *signal,float *new_signal, int m, int points)
{
float top,left,right,epsilon;
int elements;

elements=(int)pow(2.0,(double)(LEVELS-level));
/* ONE */
allocate_grid_triad(l, element);
/* TWO */
/************** ADDED THIS SECTION *******************/
if(element->up_ptr==NULL){
top=compute_iproduct(&wl[level],&signal[m*N_DELTA_T+points],elements);
element->iproduct_val=top;
element->input=top;
regenerate_signal(new_signal,&wl[level],elements,points,m,top);
}
/* top=compute_iproduct(&wl[level],&signal[m*N_DELTA_T+points],elements);*/
top=element->iproduct_val;
/******************************************************/
element->iproduct_val=top;
left=compute_iproduct(&wl[level+1],&signal[m*N_DELTA_T+points],elements/2);
element->lleg_ptr->iproduct_val=left;
element->lleg_ptr->input=left;
right=compute_iproduct(&wl[level+1],
&signal[m*N_DELTA_T+points+elements/2],elements/2);
element->rleg_ptr->iproduct_val=right;
element->rleg_ptr->input=right;
/* printf("ip=%5.3g i=%5.3g lip=%5.3g li=%5.3g rip=%5.3g ri%5.3g\n\n",
element->iproduct_val,element->input,
element->lleg_ptr->iproduct_val,element->lleg_ptr->input,
element->rleg_ptr->iproduct_val,element->rleg_ptr->input);*/
/************** ADDED THIS SECTION *******************/
regenerate_signal(new_signal,&wl[level+1],
elements/2,points,m,left);
regenerate_signal(new_signal,&wl[level+1],
elements/2,points+elements/2,m,right);
/******************************************************/
   if(level==LEVELS-2){
   inner_loop(level,l,element,wl,signal,new_signal,m,points);
   }
   else {
   return compute_trans(++level,l,element=element->lleg_ptr,wl,signal,
    new_signal,m,points);
   }

}

/************************* get_wavelets() ********************************/
/* This procedure creates the wavelet function for each resolution used */
/*************************************************************************/
void get_wavelets(int level, wavelets *w)
{
int l,elements;
FILE *fl;

/* printf("\nCreating wavelets at resolution %d\n",level);*/
elements=(int)pow(2.0,(double)(LEVELS-level));
/* printf("No. of elements is %d\n",elements);*/
MALLOC(w->wavelet,elements,float,"allocating wavelets");
```

```
loopl(elements){
if(l<elements/2)
w->wavelet[l]=(float)sqrt((pow(2.0,(double)level)));
else
w->wavelet[l]=-(float)sqrt((pow(2.0,(double)level)));
}
/* loopl(elements){
w->wavelet[l]=(float)(sqrt(pow(2.0,(double)level))*sqrt(2.0)*
sin(2.0*PI*pow(2.0,(double)level)*
(double)l/(double)N_DELTA_T));
}
if ((fl = fopen("wavelet.dat","a")) == NULL) {
printf("\n*** Can't create %s ***","wavelet.dat");
exit(0);
}
loopl(elements)
printf("%f\n",w->wavelet[l]);
fprintf(fl,"\n");
fclose(fl);*/
}

/************ compute_iproduct *********************/
/* This function produces the inner product of the signal
samples at a particular time shift and resolution*/
/**************************************************/
float compute_iproduct(wavelets *w, float *signal, int elements)
{

int i;
float val;

val=0;
loopi(elements)
val +=w->wavelet[i]*signal[i];
val /=N_DELTA_T;
return val;


}

float squash(float val, float oldmax, float oldmin, float newmax, float newmin)
{
float answer;

answer=(((val-oldmin)/(oldmax-oldmin))*(newmax-newmin)+newmin;
return answer;
}

void compute_activity(grid_element *element, int level,int t, int which)
{
  float a,exc,inh;
char string[10];
grid_element *left,*right;
int l;

a=element->activity;
inh=0;
exc=element->output+ABS2(element->input);
left=right=element;
if(level>which)
loopl(level/2){
if(left->l_ptr!=NULL){
left=left->l_ptr;
inh +=left->output;
}
```

```
if(right->r_ptr!=NULL){
right=right->r_ptr;
inh +=right->output;
}
}

element->activity=a+TAU*EPS*(-a+(MAX-a)*exc-(MIN+a)*inh*COMPGAIN);
/* printf("activity=%5.3g\n\n",element->activity);
printf("Hit enter to continue");
gets(string);*/
}

void compute_output(grid_element *element)
{
float a;

a=element->activity;
/* if(a<0)
element->output=0;
else
if(a>=0&&a<=1)
element->output=a;
else
element->output=1;*/
element->output=(a-.0009<0)?0:(SIGMOID(a-.0009));
}

main (int argc, char **argv)
{
short *temp,*temp2; /* integer representation of original data */
list *grid_list;
grid_element *node,*current_node,*top_node;
wavelets *wavelet_list;
int level,l,m,n,i;
int tot_points, points,elements;
float signal[TOT];
float newsignal[TOT],shortsignal[TOT];
float max,min;
size_t s_len;
FILE *f1,*f2,*f3,*f4,*f5;
char string[10];

mySound=[Sound new];
/* get signal, temp has integer representation of signal */
get_data(&temp,&temp2,argv[1]);
printf("Got data successfully!\n");
loopm(TOT)
signal[m]=(float)temp[m];
if ((f1 = fopen("signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f1,"%f\n",signal[m]);
grid_list=make_grid();
MALLOC(node,M_TOTAL_DTS,grid_element,"allocating first node");
MALLOC(wavelet_list,LEVELS,wavelets,"allocating wavelet list");
loopl(LEVELS)
get_wavelets(l,&wavelet_list[l]);
/* printf("Going into main loop\n"); */
loopm(M_TOTAL_DTS){
allocate_node(grid_list,&node[m]);
/* printf("\ntime is %d going into compute_trans\n",m); */
level=0;
tot_points=0;
```

```
compute_trans(level,grid_list,&node[m],wavelet_list,signal,
newsignal,m,tot_points);
}
printf("starting squash\n");
/**** This loop finds max at each time slice and uses max to squash ******/
loopm(M_TOTAL_DTS){
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
max=min=0;
   /* this loop checks for max value in time slice */
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
if(current_node->input>max)
max=current_node->input;
if(current_node->input<min)
min=current_node->input;
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}
}
/* end of check max loop */
if(max<ABS2(min))
max=ABS2(min);
printf("Max=%f min=%f at time %d---Starting squash...\n",max,min,m);
/* this loop squashes all values between using max as old max */
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
current_node->input=squash(current_node->input,max,-max,1.0,-1.0);
/* printf("after squashing: iproduct=%5.3g input=%5.3g\n\n",
current_node->iproduct_val,current_node->input);
printf("Press enter to continue!");
gets(string);*/
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}
}
 /* end of squash loop */
}
printf("end of squash loop/n");
/************************************************************************/
/************** the following computes lateral inhibition ****************/
loopm(M_TOTAL_DTS){
printf("starting LIN of time %d\n",m);
loopi(50){
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
```

```
loopl(LEVELS){
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
compute_activity(current_node,l,i,atoi(argv[2]));
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}
}
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
loopl(LEVELS){
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
compute_output(current_node);
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}

}

}
}
/***************** end of lateral inhibition *********************************/
/************ the following regenerates signal from compressed grid *******/
loopm(M_TOTAL_DTS){
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
loopl(LEVELS){
if(m==0&&l==0){
printf("top=%5.3g lleg=%5.3g rleg=%5.3g\n",
current_node->iproduct_val,current_node->lleg_ptr->iproduct_val,
current_node->rleg_ptr->iproduct_val);
printf("newtop=%5.3g newlleg=%f5.3g newrleg=%f.3g\n",
current_node->output,current_node->lleg_ptr->output,
current_node->rleg_ptr->output);
}
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
if(current_node->output!=0)
regenerate_signal(shortsignal,&wavelet_list[l],
elements,points,m,current_node->iproduct_val);
else
regenerate_signal(shortsignal,&wavelet_list[l],
elements,points,m,0);
points +=elements;
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}
}
}
printf("regenerated short signal\n");
/******************************************************************************/
/********* the following outputs the grid and compressed grid outputs *******/
if ((f3 = fopen("grid.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","grid.dat");
exit(0);
}
if ((f4 = fopen("compressed.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","compressed.dat");
exit(0);
}
loopm(M_TOTAL_DTS){
top_node=&node[m];
current_node=top_node;
grid_list->current=top_node;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-l));
loopn(N_DELTA_T/elements){
if(current_node->output!=0)
fprintf(f4,"%5.3g ",current_node->iproduct_val);
else
fprintf(f4,"%5.3g ",0);
fprintf(f3,"%5.3g ",current_node->iproduct_val);
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
fprintf(f3,"\n");
fprintf(f4,"\n");
if(grid_list->current->lleg_ptr != NULL){
grid_list->current=grid_list->current->lleg_ptr;
current_node=grid_list->current;
}
}
}
printf("outputted grid and compressed grid data\n");
/******************************************************************************/
if ((f2 = fopen("new_signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","new_signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f2,"%f\n",newsignal[m]);
if ((f5 = fopen("short_signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","short_signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f5,"%f\n",shortsignal[m]);

loopi(TOT){
temp[i]=(short)newsignal[i];
temp2[i]=(short)shortsignal[i];
}
/* get length of input file not counting null terminator */
s_len=strlen(argv[1]);
/* start name of newsoundfile with "new_" */
strcpy(newsoundfile,"new_");
/* start name of shortsoundfile with "short_" */
strcpy(shortsoundfile,"short_");
/* concatenate input file name to "new_" in newsoundfile[] */
```

```
strcat(newsoundfile,argv[1]); /* newsound[]=new_filename.snd */
/* concatenate input file name to "short_" in shortsoundfile[] */
strcat(shortsoundfile,argv[1]); /* shortsound[]=short_filename.snd */
SNDWriteSoundfile(newsoundfile,convertStruct);
SNDWriteSoundfile(shortsoundfile,shortStruct);

fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);

}
```

Appendix F. *Code for Morlet Wavelet Decomposition, Compression, and Reconstruction*

```
/************************     morlet_wavelet.m     ************************/
/*               Written by Rick Ricart, Capt, USAF                         */
/* This program computes the Morlet wavelet decomposition,compression,     */
/* and reconstruction of a time signal                                      */
/**************************************************************************/


#import <sound/sound.h>
#import <math.h>
#import <dsp/arrayproc.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import <objc/objc.h>
#import <soundkit/Sound.h>
#import <soundkit/soundkit.h>
#import <string.h>
#import <macros.h>

#define N_DELTA_T 256
#define K_TOTAL_DFS 128
#define M_TOTAL_DTS 31
#define TOT M_TOTAL_DTS*N_DELTA_T  /* approximately one sec of sound */
#define PI 3.141592654
#define LEVELS 8
#define OVERLAP 2
#define MAX 1
#define MIN 1
#define COMPGAIN 10.0
#define F0 0.25
#define TAU 0.1
#define EPS 1.0

#define ABS(x) sqrt(pow(x,2.0))
#define ABS2(x) ((float)sqrt(((double)x)*((double)x)))
#define SQUARE(x) ((x)*(x))
#define SIGMOID(x) (SQUARE(x))/(SQUARE(x)+F0)


/* Global Variables */
id mySound, newSound;
SNDSoundStruct *soundStruct, *convertStruct, *shortStruct;

/* the filenames */
char newsoundfile[80],shortsoundfile[80];

void compute_trans();
float compute_iproduct();

typedef struct grid_element {
struct grid_element *r_ptr;
struct grid_element *l_ptr;
struct grid_element *rleg_ptr;
struct grid_element *lleg_ptr;
struct grid_element *up_ptr;
```

```
float iproduct_val;
float input;
float activity;
float output;
int type;
} grid_element;

typedef struct {
grid_element *head, *current;
} list;

typedef struct {
float *wavelet;
} wavelets;

/******************** get_data() ************************************/
/* This procedure opens and reads the signal values of a sound file */
/*****************************************************************/
void get_data(short **temp,short **temp2,char *infile)
{
int error, data_size,i;
BOOL edit;
/*
    inPtr  = (short *) ((char *) inputSound   + inputSound->dataLocation);
    outPtr = (short *) ((char *) *outputSound + (*outputSound)->dataLocation);
*/
error = [mySound readSoundfile:infile];
/* initialize mySound to infile's mySound object */
soundStruct =[mySound soundStruct];
[mySound isEditable];
data_size = soundStruct->dataSize;
SNDAlloc(&convertStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
SNDAlloc(&shortStruct,data_size,SND_FORMAT_LINEAR_16,SND_RATE_CODEC,
soundStruct->channelCount,"");
SNDConvertSound(soundStruct,&convertStruct);
SNDConvertSound(soundStruct,&shortStruct);
*temp= (short *)(convertStruct+convertStruct->dataLocation);
*temp2= (short *)(shortStruct+shortStruct->dataLocation);
}

list *make_grid()
{
list *temp;

MALLOC(temp,1,list,"Allocating list");
temp->head=NULL;
temp->current=NULL;
/* printf("Made make_grid successfully\n");*/
return temp;
}

void allocate_node(list *l, grid_element *node)
{
if(l->head==NULL){
l->head=node;
l->current=node;
node->l_ptr=node->r_ptr=NULL;
}
else{
node->l_ptr=l->current;
l->current->r_ptr=node;
l->current=node;
}
node->rleg_ptr=node->lleg_ptr=NULL;
```

```
node->up_ptr=NULL;
}

void allocate_grid_triad(list *l, grid_element *element)
{
grid_element *temp1,*temp2;

/* printf("\nAllocating a triad\n");*/
/* allocate both legs */
MALLOC(temp1,1,grid_element,"allocating lleg");
MALLOC(temp2,1,grid_element,"allocating rleg");
temp1->type=0;
temp2->type=1;
element->lleg_ptr=temp1;
element->rleg_ptr=temp2;
temp1->r_ptr=temp2;
temp2->l_ptr=temp1;
temp1->up_ptr=temp2->up_ptr=element;
if(element->l_ptr==NULL||element->l_ptr->rleg_ptr==NULL)
temp1->l_ptr=NULL;
else {
temp1->l_ptr=element->l_ptr->rleg_ptr;
element->l_ptr->rleg_ptr->r_ptr=temp1;
}
temp2->r_ptr=NULL;
temp1->rleg_ptr=temp2->rleg_ptr=temp1->lleg_ptr=temp2->lleg_ptr=NULL;
l->current=element;
/* printf("allocated grid triad successfully\n");*/
}

void regenerate_signal(float *new_signal, wavelets *wl,int elements,
int points, int m, float mag)
{
int i,tot;

if(OVERLAP!=0)
tot=elements*2*OVERLAP;
else
tot=elements;
loopi(tot){
new_signal[m*N_DELTA_T+i+points-tot/2] +=wl->wavelet[i]*mag;
/* printf("%5.3g\n",new_signal[m*N_DELTA_T+i+points]);*/
}
}


void inner_loop(int level, list *l, grid_element *element, wavelets *wl,
float *signal,float *new_signal,
int m, int points)
{
int elements;

elements=(int)pow(2.0,(double)(LEVELS-level));
if(level==0)
return;
else
if(element->r_ptr==NULL){
points -=elements;
/* printf("Moving up a level, moving element ptr up, and inner loop.\n");*/
return inner_loop(--level,l,element=element->up_ptr,
wl,signal,new_signal,m,points);
}
else{
/* printf("Moving element ptr right and allocating another triad\n");*/
points +=elements;
compute_trans(level,l,element=element->r_ptr,wl,signal,
new_signal,m,points);
}
}


/******************* compute_trans() ***************************/

/****************************************************************/

void compute_trans(int level, list *l, grid_element *element, wavelets *wl,
float *signal,float *new_signal, int m, int points)
{
float top,left,right,epsilon;
int elements;

elements=(int)pow(2.0,(double)(LEVELS-level));
/* ONE */
allocate_grid_triad(l, element);
/* TWO */
/************** ADDED THIS SECTION ******************/
if(element->up_ptr==NULL){
top=compute_iproduct(&wl[level],&signal[m*N_DELTA_T+points],elements);
element->iproduct_val=top;
element->input=top;
regenerate_signal(new_signal,&wl[level],elements,points,m,top);
}
/* top=compute_iproduct(&wl[level],&signal[m*N_DELTA_T+points],elements);*/
top=element->iproduct_val;
/****************************************************/
element->iproduct_val=top;
left=compute_iproduct(&wl[level+1],&signal[m*N_DELTA_T+points],elements/2);
element->lleg_ptr->iproduct_val=left;
element->lleg_ptr->input=left;
right=compute_iproduct(&wl[level+1],
&signal[m*N_DELTA_T+points+elements/2],elements/2);
element->rleg_ptr->iproduct_val=right;
element->rleg_ptr->input=right;
/* printf("ip=%5.3g i=%5.3g lip=%5.3g li=%5.3g rip=%5.3g ri%5.3g\n\n",
element->iproduct_val,element->input,
element->lleg_ptr->iproduct_val,element->lleg_ptr->input,
element->rleg_ptr->iproduct_val,element->rleg_ptr->input);*/
/************** ADDED THIS SECTION ******************/
regenerate_signal(new_signal,&wl[level+1],
elements/2,points,m,left);
regenerate_signal(new_signal,&wl[level+1],
elements/2,points+elements/2,m,right);
/****************************************************/
   if(level==LEVELS-2){
   inner_loop(level,l,element,wl,signal,new_signal,m,points);
   }
   else {
   return compute_trans(++level,l,element=element->lleg_ptr,wl,signal,
    new_signal,m,points);
   }

}

/*********************** get_wavelets() ******************************/
/* This procedure creates the wavelet function for each resolution used */
/***********************************************************************/
void get_wavelets(int level, wavelets *w, wavelets *iw)
{
int l,elements,tot;
double time,omega,cc;
```

```
FILE *f1,*f2;

/*  These are Morlet wavelet parameters */
printf("\nCreating wavelets at resolution %d\n",level);
elements=(int)pow(2.0,(double)(LEVELS-level));
if(OVERLAP!=0)
tot=elements*2*OVERLAP;
else
tot=elements;
/* printf("No. of elements is %d\n",elements);*/
MALLOC(w->wavelet,tot,float,"allocating wavelets");
MALLOC(iw->wavelet,tot,float,"allocating wavelets");
omega=PI*sqrt(2.0/log(2.0));
cc=exp(-pow(omega,2.0)/2.0);
loopl(tot){
time=pow(2.0,(double)level)*(double)(l-tot/2)/(double)N_DELTA_T;
w->wavelet[l]=(float)((cos(omega*time)-cc)*
exp(-PI*pow(time,2.0))*
sqrt((pow(2.0,(double)level))));
  iw->wavelet[l]=(float)((sin(omega*time)-cc)*
exp(-PI*pow(time,2.0))*
sqrt((pow(2.0,(double)level))));
}

if ((f1 = fopen("rwavelet.dat","a")) == NULL) {
printf("\n*** Can't create %s ***","rwavelet.dat");
exit(0);
}
loopl(tot)
fprintf(f1,"%f\n",w->wavelet[l]);
fprintf(f1,"\n");
fclose(f1);
if ((f2 = fopen("iwavelet.dat","a")) == NULL) {
printf("\n*** Can't create %s ***","iwavelet.dat");
exit(0);
}
loopl(tot)
fprintf(f2,"%f\n",iw->wavelet[l]);
fprintf(f2,"\n");
fclose(f2);
}

/************  compute iproduct  **********************/
/* This function produces the inner product of the signal
samples at a particular time shift and resolution*/
/****************************************************/
float compute_iproduct(wavelets *w, float *signal, int elements)
{

int i,tot;
float val;

val=0;
if(OVERLAP!=0)
tot=elements*2*OVERLAP;
else
tot=elements;
loopi(tot)
val +=w->wavelet[i]*signal[i-tot/2];
val /=N_DELTA_T;
return val;
}

float squash(float val, float oldmax, float oldmin, float newmax, float newmin)
{
```

```
float answer;

answer=(((val-oldmin)/(oldmax-oldmin))*(newmax-newmin))+newmin;
return answer;
}

void compute_activity(grid_element *element, int level,int t, int which)
{
  float a,exc,inh;
char string[10];
grid_element *left,*right;
int l;

a=element->activity;
inh=0;
exc=element->output+element->input;
left=right=element;
if(level>which)
loopl(level/2){
if(left->l_ptr!=NULL){
left=left->l_ptr;
inh +=left->output;
}
if(right->r_ptr!=NULL){
right=right->r_ptr;
inh +=right->output;
}
}

element->activity=a+TAU*EPS*(-a+(MAX-a)*exc-(MIN+a)*inh*COMPGAIN);
/* printf("activity=%5.3g\n\n",element->activity);
printf("Hit enter to continue");
gets(string);*/
}

void compute_output(grid_element *element)
{
float a;

a=element->activity;
/* if(a<0)
element->output=0;
else
if(a>=0&&a<=1)
element->output=a;
else
element->output=1;*/
element->output=(a-.0009<0)?0:(SIGMOID(a-.0009));
}

main (int argc, char **argv)
{
short *temp,*temp2; /* integer representation of original data */
list *rgrid_list,*igrid_list;
grid_element *node,*current_node,*top_node;
grid_element *inode,*icurrent_node,*itop_node; /* added this */
wavelets *wavelet_list,*iwavelet_list; /*added iwavelet_list */
int level,l,m,n,i;
int tot_points, points,elements;
float signal[TOT+OVERLAP*2*N_DELTA_T];
float newsignal[TOT+OVERLAP*2*N_DELTA_T];
float shortsignal[TOT+OVERLAP*2*N_DELTA_T];
float max,min,energy_tot,energy_trunc;
size_t s_len;
FILE *f1,*f2,*f3,*f4,*f5;
```

```
char string[10];

mySound=[Sound new];
/* get signal, temp has integer representation of signal */
get_data(&temp,&temp2,argv[1]);
printf("Got data successfully!\n");
loopm(TOT)
signal[m+OVERLAP*N_DELTA_T]=(float)temp[m];
if ((f1 = fopen("signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f1,"%f\n",signal[m+OVERLAP*N_DELTA_T]);
rgrid_list=make_grid();
igrid_list=make_grid();
MALLOC(node,M_TOTAL_DTS,grid_element,"allocating first layer real nodes");
MALLOC(inode,M_TOTAL_DTS,grid_element,"allocating first layer imag node");
MALLOC(wavelet_list,LEVELS,wavelets,"allocating real wavelet list");
MALLOC(iwavelet_list,LEVELS,wavelets,"allocating imag wavelet list");
loopl(LEVELS)
get_wavelets(l,&wavelet_list[l],&iwavelet_list[l]);
printf("Going into main loop\n");
/* This loop produces the inner products of the real part of the wavelet */
loopm(M_TOTAL_DTS){
allocate_node(rgrid_list,&node[m]);
/* printf("\ntime is %d going into compute_trans\n",m); */
level=0;
tot_points=0;
compute_trans(level,rgrid_list,&node[m],wavelet_list,
&signal[OVERLAP*N_DELTA_T],&newsignal[OVERLAP*N_DELTA_T],
m,tot_points);
}
/* This loop produces the inner products of the imag. part of the wavelet */
loopm(M_TOTAL_DTS){
allocate_node(igrid_list,&inode[m]);
level=0;
tot_points=0;
compute_trans(level,igrid_list,&inode[m],iwavelet_list,
&signal[OVERLAP*N_DELTA_T],&newsignal[OVERLAP*N_DELTA_T],
m,tot_points);
}
printf("starting squash\n");
/**** This loop finds max at each time slice and uses max to squash ******/
loopm(M_TOTAL_DTS){
top_node=&node[m];
itop_node=&inode[m];
current_node=top_node;
icurrent_node=top_node;
rgrid_list->current=top_node;
igrid_list->current=itop_node;
max=0;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
if((current_node->input=
(float)sqrt(pow((double)current_node->input,2.0)+
pow((double)icurrent_node->input,2.0)))>max)
max=current_node->input;
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
icurrent_node=icurrent_node->r_ptr;
}
}
```

```
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
igrid_list->current=igrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
icurrent_node=igrid_list->current;
}
}
printf("Max=%f min=%f at time %d---Starting squash...\n",max,min,m);
top_node=&node[m];
current_node=top_node;
rgrid_list->current=top_node;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
current_node->input=squash(current_node->input,max,0,1.0,0);
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
}
}
}
printf("end of squash loop\n");
/*************************************************************************/
/************** the following computes lateral inhibition *****************/
loopm(M_TOTAL_DTS){
printf("\nstarting LIN at time %d\n",m);
loopi(50){
top_node=&node[m];
current_node=top_node;
rgrid_list->current=top_node;
loopl(LEVELS){
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
compute_activity(current_node,l,i,atoi(argv[2]));
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
}
}
top_node=&node[m];
current_node=top_node;
rgrid_list->current=top_node;
loopl(LEVELS){
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
compute_output(current_node);
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
}

}
```

```
}
}
/***************** end of lateral inhibition ********************************/
/************** the following energy normalizes *****************/
printf("\nstarting energy normalization\n",m);
loopm(M_TOTAL_DTS){
top_node=&node[m];
itop_node=&inode[m];
current_node=top_node;
icurrent_node=top_node;
rgrid_list->current=top_node;
igrid_list->current=itop_node;
/** find the total energy and truncated energy at each resolution **/
   loopl(LEVELS){
   elements=(int)pow(2.0,(double)(LEVELS-1));
   loopn(N_DELTA_T/elements){
   energy_tot += (float)pow((double)current_node->input,2.0);
   if(current_node->output!=0)
   energy_trunc += (float)pow((double)current_node->input,2.0);
if(current_node->r_ptr != NULL)
current_node=current_node->r_ptr;
   }
current_node=rgrid_list->current;
/** multiply both the real and imag coefficients by the energy change **/
   loopn(N_DELTA_T/elements){
   current_node->iproduct_val *= energy_tot/energy_trunc;
icurrent_node->iproduct_val *= energy_tot/energy_trunc;
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
icurrent_node=icurrent_node->r_ptr;
}
}
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
igrid_list->current=igrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
icurrent_node=igrid_list->current;
}
  }
}
/***************** end of energy normalization **************************/

/************ the following regenerates signal from compressed grid *******/
printf("\nstarting to regenerate signals\n");
loopm(M_TOTAL_DTS){
top_node=&node[m];
itop_node=&inode[m];
current_node=top_node;
icurrent_node=top_node;
rgrid_list->current=top_node;
igrid_list->current=itop_node;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
if(current_node->output!=0){
regenerate_signal(&shortsignal[OVERLAP*N_DELTA_T],
&wavelet_list[l],
elements,points,m,current_node->iproduct_val);
regenerate_signal(&shortsignal[OVERLAP*N_DELTA_T],
&iwavelet_list[l],
elements,points,m,icurrent_node->iproduct_val);
}
else{
```

```
regenerate_signal(&shortsignal[OVERLAP*N_DELTA_T],
&wavelet_list[1],
elements,points,m,0);
regenerate_signal(&shortsignal[OVERLAP*N_DELTA_T],
&iwavelet_list[1],
elements,points,m,0);
}
points +=elements;
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
icurrent_node=icurrent_node->r_ptr;
}
}
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
igrid_list->current=igrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
icurrent_node=igrid_list->current;
}
}
}
printf("regenerated short signal\n");
/*****************************************************************************/
/********** the following outputs the grid and compressed grid outputs *******/
if ((f3 = fopen("grid.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","grid.dat");
exit(0);
}
if ((f4 = fopen("compressed.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","compressed.dat");
exit(0);
}
loopm(M_TOTAL_DTS){
top_node=&node[m];
current_node=top_node;
rgrid_list->current=top_node;
loopl(LEVELS){
points=0;
elements=(int)pow(2.0,(double)(LEVELS-1));
loopn(N_DELTA_T/elements){
if(current_node->output!=0) /*change this if */
fprintf(f4,"%5.3g ",current_node->input);
else
fprintf(f4,"%5.3g ",0);
fprintf(f3,"%5.3g ",current_node->input);
if(current_node->r_ptr != NULL){
current_node=current_node->r_ptr;
}
}
fprintf(f3,"\n");
fprintf(f4,"\n");
if(rgrid_list->current->lleg_ptr != NULL){
rgrid_list->current=rgrid_list->current->lleg_ptr;
current_node=rgrid_list->current;
}
}
}
printf("outputted grid and compressed grid data\n");
/*****************************************************************************/
if ((f2 = fopen("new_signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","new_signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f2,"%f\n",newsignal[m+OVERLAP*N_DELTA_T]);
```

```
if ((f5 = fopen("short_signal.dat","w")) == NULL) {
printf("\n*** Can't create %s ***","short_signal.dat");
exit(0);
}
loopm(TOT)
fprintf(f5,"%f\n",shortsignal[m+OVERLAP*N_DELTA_T]);

loopi(TOT){
temp[i]=(short)newsignal[i+OVERLAP*N_DELTA_T];
temp2[i]=(short)shortsignal[i+OVERLAP*N_DELTA_T];
}
/* get length of input file not counting null terminator */
s_len=strlen(argv[1]);
/* start name of newsoundfile with "new_" */
strcpy(newsoundfile,"new_");
/* start name of shortsoundfile with "short_" */
strcpy(shortsoundfile,"short_");
/* concatenate input file name to "new_" in newsoundfile[] */
strcat(newsoundfile,argv[1]); /* newsound[]=new_filename.snd */
/* concatenate input file name to "short_" in shortsoundfile[] */
strcat(shortsoundfile,argv[1]); /* shortsound[]=short_filename.snd */
SNDWriteSoundfile(newsoundfile,convertStruct);
SNDWriteSoundfile(shortsoundfile,shortStruct);

fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);

}
```

Bibliography

1. Alenquer, Luis M. F. *Rule Based Sinusoidal Encoding of Speech.* MS thesis, AFIT/GE/ENG/90M-1, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1990.
2. Ashmore, Jonathan F. "The Cellular Physiology of Isolated Outer Hair Cells: Implications for Cochlear Frequency Selectivity." In Moore, Brian C.J. and Roy D. Patterson, editors, *Auditory Frequency Selectivity*, pages 103–108, New York: Plenum Press, 1986.
3. Aware, Inc. *Develop, Apply, and Evaluate Wavelet Technology.* Quarterly Technical Report AD900302, Cambridge, MA: Aware, Inc., March 1990.
4. Bashir, Nadeem A. *Phoneme Adjustment in Enhanced Speech.* MS thesis, AFIT/ENG/89M-2, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1989 (AD–A206 357).
5. Bastiaans, Martin J. "Gabor's expansion of a signal into Gaussian elementary signals," *Proceedings of the IEEE, 68*(4):538–539 (April 1980).
6. Bastiaans, Martin J. "A sampling theorem for the complex spectrogram, and Gabor's expansion of a signal in Gaussian elementary signals," *Optical Engineering, 20*(4):594–598 (July/August 1981).
7. Bovik, Alan Conrad, et al. "Multichannel texture analysis using localized spatial filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*(1):55–73 (January 1990).
8. Brownell, William E. "Outer Hair Cell Motility and Cochlear Frequency Selectivity." In Moore, Brian C.J. and Roy D. Patterson, editors, *Auditory Frequency Selectivity*, pages 109–118, New York: Plenum Press, 1986.
9. Claasen, T.A.C.M and W.F.G. Mecklenbräuker. "The Wigner distribution—a tool for time–frequency signal analysis, Part I: Continuous-time signals," *Philips Journal of Research, 35*:217–250 (1980).
10. Claasen, T.A.C.M and W.F.G. Mecklenbräuker. "The Wigner distribution—a tool for time–frequency signal analysis, Part II: Discrete-time signals," *Philips Journal of Research, 35*:276–300 (1980).
11. Claasen, T.A.C.M and W.F.G. Mecklenbräuker. "The Wigner distribution—a tool for time–frequency signal analysis, Part III: Relations with other time-frequency signal transformations," *Philips Journal of Research, 35*:372–389 (1980).
12. Coifman, R.R. "Wavelet Analysis and Signal Processing." In Auslander, L., et al., editors, *Signal Processing, Part I: Signal Processing Theory*, pages 59–73, New York: Springer-Verlag, 1990.

13. Daubechies, Ingrid. "Orthogonal bases of compactly supported wavelets," *Communication on Pure and Applied Mathematics, XLI*:909–996 (1988).

14. Daubechies, Ingrid. "The wavelet transform, time-frequency localization, and signal analysis," *IEEE Transactions on Information Theory, 36*(5):961–1005 (September 1990).

15. Daubechies, Ingrid, et al. "Painless nonorthogonal expansions," *Journal of Mathematical Physics, 27*:1271–1283 (1986).

16. Daugman, John G. "Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 36*(7):1169–1179 (July 1988).

17. Dudley, Homer. "Remaking speech," *Journal of the Acoustical Society of America, 11*:169–177 (1939).

18. Duffin, R.J. and A.C. Schaeffer. "A class of nonharmonic Fourier series," *Transactions of the American Mathematical Society, 72*:1271–1283 (1952).

19. Einziger, P.D. "Numerical implementation of the Gabor representation," *Electronic Letters, 24*:810–811 (1988).

20. Ellias, Samuel A. and Stephen Grossberg. "Pattern formation, contrast control, and oscillations in the short term memory of shunting on-center off-surround networks," *Biological Cybernetics, 20*:69–98 (1975).

21. Fastl, Hugo and Edwin Schorer. "Critical Bandwidth at Low Frequencies Reconsidered." In Moore, Brian C.J. and Roy D. Patterson, editors, *Auditory Frequency Selectivity*, pages 311–322, New York: Plenum Press, 1986.

22. Friedlander, Benjamin and Boaz Porat. "Detection of transient signals by the Gabor representation," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 37*(2):169–180 (February 1989).

23. Gabor, D. "Theory of communication," *The Journal of the Institution of Electrical Engineers, 93*:429–457 (1946).

24. Gabor, D. "New possibilities in speech transmission," *The Journal of the Institution of Electrical Engineers, 94*:369–390 (1947).

25. Grossberg, Stephen. "Contour enhancement, short term memory, and constancies in reverberating neural networks," *Studies in Applied Mathematics, 52*:217–257 (1973).

26. Grossberg, Stephen. "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks, 1*(1):17–61 (1988).

27. Grossberg, Stephen and Daniel Levine. "Some development and attentional biases in the contrast enhancement and short term memory of recurrent neural networks," *Journal of Theoretical Biology, 53*:341–380 (1975).

28. Grossman, A. and J. Morlet. "Decomposition of functions into wavelets of constant shape, and related transforms." In *Mathematics and Physics, Lectures on Recent Results*, pages 135–165, Singapore: World Scienfific, 1985.

29. Gulick, W. Lawrence, et al. *Hearing: Physiological Acoustics, Neural Coding, and Psychoacoustics*. New York: Oxford University Press, 1989.

30. Harris, Fredric J. "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, *66*(1):51–83 (January 1978).

31. Heil, Christopher. "Wavelets and Frames." In Auslander, L., et al., editors, *Signal Processing, Part I: Signal Processing Theory*, pages 147–160, New York: Springer-Verlag, 1990.

32. Heil, Christopher E. and David F. Walnut. "Continuous and discrete wavelet transforms," *SIAM Review*, *31*(4):628–666 (December 1989).

33. Helstrom, Carl W. "An expansion of a signal in Gaussian elementary signals," *IEEE Transactions on Information Theory*, *IT–12*:81–82 (January 1966).

34. Hodgkin, A.L. and A.F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Journal of Physiology*, *117*:500–544 (1952).

35. Jacobson, D. and Harry Wechsler. "Joint spatial/spatial-frequency representation," *Signal Processing*, *14*:37–68 (1988).

36. Janssen, A.J.E.M. "Gabor representation of generalized functions," *Journal of Mathematical Analysis and Applications*, *83*:377–394 (1981).

37. Janssen, A.J.E.M. "The Zak transform: A signal transform for sampled time-continuous signals," *Philips Journal of Research*, *43*:23–69 (1988).

38. Kabrisky, Matthew, et al. "Reconstruction of Mutilated Speech," *IEEE Aerospace and Electronics Systems Magazine*, pages 39–43 (September 1989).

39. Kim, D.O. "Functional Roles of the Inner and Outer-Hair-Cell Subsystems in the Cochlea and Brainstem." In Berlin, Charles I., editor, *Heaing Science: Recent Advances*, chapter 7, pages 241–262, San Diego: College Hill Press, 1984.

40. Knudsen, Eric I. "Center-surround organization of auditory receptive fields in the owl," *Science*, *202*(17):778–780 (November 1978).

41. Lazzaro, J., et al. "Winner-take-all Networks of $O(N)$ Complexity." In Touretzky, David S., editor, *Advances in Neural Information Processing Systems I*, pages 703–711, San Mateo, CA: Morgan Kaufman Publishers, Inc., 1989.

42. Lippmann, Richard P. "Review of Neural Networks for Speech Recognition," *Neural Computation*, *1*:1–38 (1989).

43. Ludeman, Lonnie C. *Fundamentals of Digital Signal Processing*. New York: John Wiley and Sons Inc., 1986.

44. Mallat, Stephane G. "A theory for multiresolution signal decomposition: The wavelet transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 11*(7):674–693 (July 1989).

45. Mallat, Stephane G. and Sifen Zhong. *Complete Signal Representation With Multiscale Edges.* Technical Report 483, New York: New York University, Department of Computer Sciences, Courant Institute of Mathematical Sciences, December 1989.

46. Marr, D. and E. Hildreth. "A theory of edge detection," *Proceedings of the Royal Society of London, 207*:187–217 (1980).

47. Martens, Jean-Bernard. "The Hermite Transform–Theory," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 38*(9):1595–1606 (September 1990).

48. McAulay, R.J. and T.F. Quatieri. "Magnitude only reconstruction using a sinusoidal speech model." In *Proceedings of the International Conference of Acoustics, Speech, and Signal Processing*, pages 27.6.1–27.6.4, 1984.

49. McMillan, Vance M. *Rule-Based Frequency Domain Speech Coding.* MS thesis, AFIT/GE/ENG/90D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1990.

50. Moore, Brian C.J. and Brian R. Glasberg. "Suggested fomulae for calculating auditory-filter bandwidths and excitation patterns," *Journal of the Acoustical Society of America, 74*(3):750–753 (September 1983).

51. Morlet, J., et al. "Wave propagation and sampling theory," *Geophysics, 47*(2):203–236 (February 1982).

52. Nabet, Bahram, et al. "Analog Implementation of Shunting Neural Networks." In Touretzky, David S., editor, *Advances in Neural Information Processing Systems I*, pages 695–702, San Mateo, CA: Morgan Kaufman Publishers, Inc., 1989.

53. Parsons, Thomas W. *Voice and Speech Processing.* New York: McGraw-Hill Book Company, 1987.

54. Patterson, R.D. "Auditory filter shapes derived with noise stimuli," *Journal of the Acoustical Society of America, 59*:640–654 (1976).

55. Pisoni, David B. "Speech perception: Some new directions in research and theory," *Journal of the Acoustical Society of America, 78*:381–388 (1985).

56. Porat, Moshe and Yehoshua Y. Zeevi. "The generalized Gabor scheme of image representation in biological and machine vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 10*:452–468 (1988).

57. Quatieri, Thomas E. and R.J. McAulay. "Speech transformation based on a sinusoidal representation." In *Proceedings of the International Conference of Acoustics, Speech, and Signal Processing*, pages 13.5.1–13.5.4, 1985.

58. Rabiner, Lawrence R. and Ronald W. Schafer. *Digital Signal Processing of Speech.* Englewood Cliffs NJ: Prentice-Hall, Inc., 1978.

59. Ratliff, F. *Mach Bands: Quatitative Studies on Neural Networks in the Retina.* New York: Holden-Day, 1965.

60. Ratliff, F., et al. "Spatial and temporal aspects of retinal inhibitory interactions," *Journal of the Optical Society of America, 53*:110–120 (1963).

61. Reed, Todd R. and Harry Wechsler. "Segmentation of textured images and gestalt organization using spatial/spatial-frequency representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*(1):1–11 (January 1990).

62. Sellick, P.M and I.J. Russell. "Two-tone suppression in cochlear hair cells," *Hearing Research, 1*:227–236 (1979).

63. Shamma, Shihab A. "Speech processing in the auditory system II: Lateral inhibition and the central processing of speech evoked activity in the auditory nerve," *Journal of the Acoustical Society of America, 78*(5):1622–1632 (November 1985).

64. Shepherd, Gordon M. *Neurobiology* (Second Edition). New York: Oxford University Press, 1988.

65. Sklar, Bernard. *Digital Communications: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice Hall, 1988.

66. Strang, Gilbert. "Wavelets and dilation equations: A brief introduction," *SIAM Review, 31*(4):614–627 (December 1989).

67. Turner, M.R. "Texture discrimination by Gabor functions," *Biological Cybernetics, 55*:71–82 (1986).

68. Weyl, H. *Theory of Groups and Quantum Mechanics.* New York: Dutton, 1932.

69. Wiederhold, Michael L. "Physiology of the Olivocochlear System." In Altschuler, Richard A., et al., editors, *Neurobilogy of Hearing*, pages 349–370, New York: Raven Press, 1986.

70. Zwicker, E. and E. Terhardt. "Analytical expressions for critical-band rate and critical bandwidth as a function of frequency," *Journal of the Acoustical Society of America, 68*(5):1523–1525 (November 1980).

Vita

Captain Richard Ricart was born on February 1, 1954 in Havana, Cuba. He immigrated with his family to the United States in 1960, and took up residence in Miami, Florida. Captain Ricart graduated from Miami Coral Park Senior High School, Miami, Florida in 1972 and joined the City of Miami Fire Department in 1975, where he worked as a firefighter/paramedic until January, 1984. In June, 1984 he graduated Magna Cum Laude from the University of Miami in Coral Gables, Florida, with a B.S.E.E and soon after received his commission from Officer Training School in October 1984. His first assignment was to the Avionics Laboratory at Wright-Patterson AFB, OH, where he served as an Artificial Intelligence Systems Engineer, System Avionics Division until entering the School of Engineering, Air Force Institute of Technology in May 1989.

Permanent address: 2711 State Route 235
Xenia, Ohio 45385

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1990	3. REPORT TYPE AND DATES COVERED *Master's Thesis*

4. TITLE AND SUBTITLE
Speech Coding and Compression Using Wavelets and Lateral Inhibitory Networks

5. FUNDING NUMBERS

6. AUTHOR(S)
Richard Ricart, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/90D-51

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
RADC/IRAA, Griffiss AFB, NY 13441

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A generalized method of compressing speech using lateral inhibition networks (LINs) is proposed in this thesis. Speech signals are first decomposed using three time/frequency transforms: the short-time Fourier transform, the Gabor transform, and the affine wavelet transform. Redundant time/frequency coefficients are then automatically eliminated using the dynamics of a LIN. The speech is finally resynthesized from the compressed representations and tested for intelligibility. The LIN is modeled as a system of shunting non-linear differential equations in the form of the neuronal cell membrane equations first found by Hodgkin and Huxley. Thus, the LIN can be described as a neural network. LINs perform several functions; the most important being contrast enhancement. In contrast enhancement, spatial peaks and edges as well as temporal changes are highlighted. The best results were obtained from the compressed short-time Fourier spectrum. Nearly 30 times compression—which relates to over 95% elimination of the spectrum—resulted in clearly intelligible speech. However, elimination of only a small percentage of the affine wavelet spectrum produces wide band noise that is offensive to the hearing mechanisms.

14. SUBJECT TERMS
Speech Coding, Speech Compression, Wavelets, Lateral Inhibition Networks, Gabor Transform

15. NUMBER OF PAGES
175

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102